wxErlang

Copyright © 2009-2014 Ericsson AB. All Rights Reserved.
wxErlang 0.99.2
March 16 2014

Copyright © 2009-2014 Ericsson AB. All Rights Reserved.
The contents of this file are subject to the Erlang Public License, Version 1.1, (the "License"); you may not use this file except in compliance with the License. You should have received a copy of the Erlang Public License along with this software. If not, it can be retrieved online at http://www.erlang.org/. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. Ericsson AB. All Rights Reserved..

March 16 2014

1 wxErlang User's Guide

The *wxErlang* application is an api for writing graphical user interfaces with wxWidgets.

1.1 wx the erlang binding of wxWidgets

The *wx* application is an erlang binding of *wxWidgets*. This document describes the erlang mapping to wxWidgets and it's implementation. It is not a complete users guide to wxWidgets. If you need that, you will have to read the wxWidgets documentation instead. *wx* tries to keep a one-to-one mapping with the original api so that the original documentation and examples shall be as easy as possible to use.

wxErlang examples and test suite can be found in the erlang src release. They can also provide some help on how to use the api.

This is currently a very brief introduction to *wx*. The application is still under development, which means the interface may change, and the test suite currently have a poor coverage ratio.

1.1.1 Contents

1.1.2 Introduction

The original *wxWidgets* is an object-oriented (C++) api and that is reflected in the erlang mapping. In most cases each class in wxWidgets is represented as a module in erlang. This gives the *wx* application a huge interface, spread over several modules, and it all starts with the *wx* module. The *wx* module contains functions to create and destroy the gui, i.e. `wx:new/0`,`wx:destroy/0`, and some other useful functions.

Objects or object references in *wx* should be seen as erlang processes rather then erlang terms. When you operate on them they can change state, e.g. they are not functional objects as erlang terms are. Each object has a type or rather a class, which is manipulated with the corresponding module or by sub-classes of that object. Type checking is done so that a module only operates on it's objects or inherited classes.

An object is created with *new* and destroyed with *destroy*. Most functions in the classes are named the same as their C++ counterpart, except that for convenience, in erlang they start with a lowercase letter and the first argument is the object reference. Optional arguments are last and expressed as tagged tuples in any order.

For example the *wxWindow* C++ class is implemented in the *wxWindow* erlang module and the member *wxWindow::CenterOnParent* is thus *wxWindow:centerOnParent*. The following C++ code:

```
wxWindow MyWin = new wxWindo();
MyWin.CenterOnParent(wxVERTICAL);
...
delete MyWin;
```

would in erlang look like:

```
MyWin = wxWindow:new(),
wxWindow:centerOnParent(MyWin, [{dir,?wxVERTICAL}]),
...
wxWindow:destroy(MyWin),
```

When you are reading wxWidgets documentation or the examples, you will notice that some of the most basic classes are missing in *wx*, they are directly mapped to corresponding erlang terms:

wxPoint is represented by {Xcoord,Ycoord}
wxSize is represented by {Width,Height}
wxRect is represented by {Xcoord,Ycoord,Width,Height}
wxColour is represented by {Red,Green,Blue[,Alpha]}
wxPoint is represented by {Xcoord,Ycoord}
wxString is represented by *unicode:charlist()*
wxGBPosition is represented by {Row,Column}
wxGBSpan is represented by {RowSpan,ColumnSPan}
wxGridCellCoords is represented by {Row,Column}

In the places where the erlang api differs from the original one it should be obvious from the erlang documentation which representation has been used. E.g. the C++ arrays and/or lists are sometimes represented as erlang lists and sometimes as tuples.

Colours are represented with {Red,Green,Blue[,Alpha]}, the Alpha value is optional when used as an argument to functions, but it will always be returned from *wx* functions.

Defines, enumerations and global variables exists in `wx.hrl` as defines. Most of these defines are constants but not all. Some are platform dependent and therefore the global variables must be instantiated during runtime. These will be acquired from the driver with a call, so not all defines can be used in matching statements. Class local enumerations will be prefixed with the class name and a underscore as in `ClassName_Enum`.

Additionally some global functions, i.e. non-class functions, exist in the `wx_misc` module.

wxErlang is implemented as a (threaded) driver and a rather direct interface to the C++ api, with the drawback that if the erlang programmer does an error, it might crash the emulator.

Since the driver is threaded it requires a *smp* enabled emulator, that provides a thread safe interface to the driver.

1.1.3 Multiple processes and memory handling

The intention is that each erlang application calls wx:new() once to setup it's gui which creates an environment and a memory mapping. To be able to use *wx* from several processes in your application, you must share the environment. You can get the active environment with `wx:get_env/0` and set it in the new processes with `wx:set_env/1`. Two processes or applications which have both called wx:new() will not be able use each others objects.

```
wx:new(),
MyWin = wxFrame:new(wx:null(), 42, "Example", []),
Env = wx:get_env(),
spawn(fun() ->
         wx:set_env(Env),
         %% Here you can do wx calls from your helper process.
         ...
      end),
...
```

When `wx:destroy/0` is invoked or when all processes in the application have died, the memory is deleted and all windows created by that application are closed.

The *wx* application never cleans or garbage collects memory as long as the user application is alive. Most of the objects are deleted when a window is closed, or at least all the objects which have a parent argument that is non null. By using `wxCLASS:destroy/1` when possible you can avoid an increasing memory usage. This is especially important when *wxWidgets* assumes or recommends that you (or rather the C++ programmer) have allocated the object on the stack since that will never be done in the erlang binding. For example `wxDC` class or its sub-classes or `wxSizerFlags`.

Currently the dialogs show modal function freezes wxWidgets until the dialog is closed. That is intended but in erlang where you can have several gui applications running at the same time it causes trouble. This will hopefully be fixed in future *wxWidgets* releases.

1.1.4 Event Handling

Event handling in *wx* differs most the from the original api. You must specify every event you want to handle in *wxWidgets*, that is the same in the erlang binding but can you choose to receive the events as messages or handle them with callback funs.

Otherwise the event subscription is handled as *wxWidgets* dynamic event-handler connection. You subscribe to events of a certain type from objects with an *ID* or within a range of ID:s. The callback fun is optional, if not supplied the event will be sent to the process that called *connect/2*. Thus, a handler is a callback fun or a process which will receive an event message.

Events are handled in order from bottom to top, in the widgets hierarchy, by the last subscribed handler first. Depending on if `wxEvent:skip()` is called the event will be handled by the other handler(s) afterwards. Most of the events have default event handler(s) installed.

Message events looks like *#wx{id=integer(), obj=wx:wxObject(), userData=term(), event=Rec* }. The *id* is the identifier of the object that received the event. The *obj* field contains the object that you used *connect* on. The *userData* field contains a user supplied term, this is an option to *connect*. And the *event* field contains a record with event type dependent information. The first element in the event record is always the type you subscribed to. For example if you subscribed to *key_up* events you will receive the `#wx{event=Event}` where *Event* will be a *wxKey* event record where `Event#wxKey.type = key_up`.

In *wxWidgets* the developer have to call `wxEvent:skip()` if he wants the event to be processed by other handlers. You can do the same in *wx* if you use callbacks. If you want the event as messages you just don't supply a callback and you can set the *skip* option in *connect* call to true or false, the default it is false. True means that you get the message but let the subsequent handlers also handle the event. If you want to change this behavior dynamically you must use callbacks and call `wxEvent:skip()`.

Callback event handling is done by using the optional callback *fun/2* when attaching the handler. The *fun(#wx{},wxObject()* must take two arguments where the first is the same as with message events described above and the second is an object reference to the actual event object. With the event object you can call `wxEvent:skip()` and access all the data. When using callbacks you must call `wxEvent:skip()` by yourself if you want any of the events to be forwarded to the following handlers. The actual event objects are deleted after the *fun* returns.

The callbacks are always invoked by another process and have exclusive usage of the gui when invoked. This means that a callback fun can not use the process dictionary and should not make calls to other processes. Calls to another process inside a callback fun may cause a deadlock if the other process is waiting on completion of his call to the gui.

1.1.5 Acknowledgments

Mats-Ola Persson wrote the initial *wxWidgets* binding as part of his master thesis. The current version is a total re-write but many ideas have been reused. The reason for the re-write was mostly due to the limited requirements he had been given by us.

Also thanks to the *wxWidgets* team that develops and supports it so we have something to use.

2 Reference Manual

The *wxErlang* application is an api for writing graphical user interfaces with wxWidgets.

WX

Erlang module

A port of **wxWidgets**.

This is the base api of **wxWidgets**. This module contains functions for starting and stopping the wx-server, as well as other utility functions.

wxWidgets is object oriented, and not functional. Thus, in wxErlang a module represents a class, and the object created by this class has an own type, wxCLASS(). This module represents the base class, and all other wxMODULE's are sub-classes of this class.

Objects of a class are created with wxCLASS:new(...) and destroyed with wxCLASS:destroy(). Member functions are called with wxCLASS:member(Object, ...) instead of as in C++ Object.member(...).

Sub class modules inherit (non static) functions from their parents. The inherited functions are not documented in the sub-classes.

This erlang port of wxWidgets tries to be a one-to-one mapping with the original wxWidgets library. Some things are different though, as the optional arguments use property lists and can be in any order. The main difference is the event handling which is different from the original library. See *wxEvtHandler*.

The following classes are implemented directly as erlang types: wxPoint={x,y},wxSize={w,h},wxRect={x,y,w,h},wxColour={r,g,b [,a]}, wxString=*unicode:chardata()*, wxGBPosition={r,c},wxGBSpan={rs,cs},wxGridCellCoords={r,c}.

wxWidgets uses a process specific environment, which is created by *wx:new/0*. To be able to use the environment from other processes, call *get_env/0* to retrieve the environment and *set_env/1* to assign the environment in the other process.

Global (classless) functions are located in the wx_misc module.

DATA TYPES

```
wx_colour() = {R::byte(), G::byte(), B::byte()} | wx_colour4()
wx_colour4() = {R::byte(), G::byte(), B::byte(), A::byte()}
wx_datetime() = {{Year::integer(), Month::integer(), Day::integer()},
{Hour::integer(), Minute::integer(), Second::integer()}}
```

In Local Timezone

```
wx_enum() = integer()
```

Constant defined in wx.hrl

```
wx_env() = #wx_env{}
```

Opaque process environment

```
wx_memory() = binary() | #wx_mem{}
```

Opaque memory reference

```
wx_object() = #wx_ref{}
```

Opaque object reference

```
wx_wxHtmlLinkInfo() = #wxHtmlLinkInfo{href=undefined | chardata() (see module
unicode), target=undefined | chardata() (see module unicode)}
wx_wxMouseState() = #wxMouseState{x=undefined | integer(), y=undefined
| integer(), leftDown=undefined | boolean(), middleDown=undefined
| boolean, rightDown=undefined | boolean, controlDown=undefined |
```

```
boolean(), shiftDown=undefined | boolean(), altDown=undefined | boolean(),
metaDown=undefined | boolean(), cmdDown=undefined | boolean()}
```

See #wxMouseState{} defined in wx.hrl

Exports

parent_class(X1) -> term()

new() -> wx_object()

Starts a wx server.

new(Option::[Option]) -> wx_object()

Types:

Option = {debug, list() | atom()}

Starts a wx server. Option may be {debug, Level}, see debug/1.

destroy() -> ok

Stops a wx server.

get_env() -> wx_env()

Gets this process's current wx environment. Can be sent to other processes to allow them use this process wx environment.

See also: set_env/1.

set_env(Wx_env::wx_env()) -> ok

Sets the process wx environment, allows this process to use another process wx environment.

null() -> wx_object()

Returns the null object

is_null(Wx_ref::wx_object()) -> boolean()

Returns true if object is null, false otherwise

getObjectType(Wx_ref::wx_object()) -> atom()

Returns the object type

typeCast(Old::wx_object(), NewType::atom()) -> wx_object()

Casts the object to class NewType. It is needed when using functions like wxWindow:findWindow/2, which returns a generic wxObject type.

batch(Fun::function()) -> term()

Batches all `wx` commands used in the fun. Improves performance of the command processing by grabbing the wxWidgets thread so that no event processing will be done before the complete batch of commands is invoked.

See also: foldl/3, foldr/3, foreach/2, map/2.

`foreach(Fun::function(), List::list()) -> ok`

Behaves like *lists:foreach/2* but batches wx commands. See *batch/1*.

`map(Fun::function(), List::list()) -> list()`

Behaves like *lists:map/2* but batches wx commands. See *batch/1*.

`foldl(Fun::function(), Acc::term(), List::list()) -> term()`

Behaves like *lists:foldl/3* but batches wx commands. See *batch/1*.

`foldr(Fun::function(), Acc::term(), List::list()) -> term()`

Behaves like *lists:foldr/3* but batches wx commands. See *batch/1*.

`create_memory(Size::integer()) -> wx_memory()`

Creates a memory area (of Size in bytes) which can be used by an external library (i.e. opengl). It is up to the client to keep a reference to this object so it does not get garbage collected by erlang while still in use by the external library.

This is far from erlang's intentional usage and can crash the erlang emulator. Use it carefully.

`get_memory_bin(Wx_mem::wx_memory()) -> binary()`

Returns the memory area as a binary.

`retain_memory(Wx_mem::wx_memory()) -> ok`

Saves the memory from deletion until release_memory/1 is called. If release_memory/1 is not called the memory will not be garbage collected.

`release_memory(Wx_mem::wx_memory()) -> ok`

`debug(List::Level | [Level]) -> ok`

Types:

`Level = none | verbose | trace | driver | integer()`

Sets debug level. If debug level is 'verbose' or 'trace' each call is printed on console. If Level is 'driver' each allocated object and deletion is printed on the console.

`demo() -> ok | {error, atom()}`

Starts a wxErlang demo if examples directory exists and is compiled

wx_object

Erlang module

wx_object - Generic wx object behaviour

This is a behaviour module that can be used for "sub classing" wx objects. It works like a regular gen_server module and creates a server per object.

NOTE: Currently no form of inheritance is implemented.

The user module should export:

init(Args) should return
{wxObject, State} | {wxObject, State, Timeout} | ignore | {stop, Reason}

handle_call(Msg, {From, Tag}, State) should return
{reply, Reply, State} | {reply, Reply, State, Timeout} | {noreply, State} | {noreply, State, Timeout} | {stop, Reason, Reply, State}

Asynchronous window event handling:
handle_event(#wx{}, State) should return
{noreply, State} | {noreply, State, Timeout} | {stop, Reason, State}

Info is message e.g. {'EXIT', P, R}, {nodedown, N}, ...
handle_info(Info, State) should return , ...
{noreply, State} | {noreply, State, Timeout} | {stop, Reason, State}

When stop is returned in one of the functions above with Reason = normal | shutdown | Term, terminate(State) is called. It lets the user module clean up, it is always called when server terminates or when wxObject() in the driver is deleted. If the Parent process terminates the Module:terminate/2 function is called.
terminate(Reason, State)

Example:

```
-module(myDialog).
-export([new/2, show/1, destroy/1]).  %% API
-export([init/1, handle_call/3, handle_event/2,
         handle_info/2, code_change/3, terminate/2]).
         new/2, showModal/1, destroy/1]).  %% Callbacks

%% Client API
new(Parent, Msg) ->
   wx_object:start(?MODULE, [Parent,Id], []).

show(Dialog) ->
   wx_object:call(Dialog, show_modal).

destroy(Dialog) ->
   wx_object:call(Dialog, destroy).

%% Server Implementation ala gen_server
init([Parent, Str]) ->
   Dialog = wxDialog:new(Parent, 42, "Testing", []),
   ...
   wxDialog:connect(Dialog, command_button_clicked),
   {Dialog, MyState}.

handle_call(show, _From, State) ->
   wxDialog:show(State#state.win),
   {reply, ok, State};
```

```
...
handle_event(#wx{}, State) ->
   io:format("Users clicked button~n",[]),
   {noreply, State};
...
```

Exports

start(Name, Mod, Args, Options) -> wxWindow() (see module wxWindow)

Types:

Name = {local, atom()}
Mod = atom()
Args = term()
Options = [{timeout, Timeout} | {debug, [Flag]}]
Flag = trace | log | {logfile, File} | statistics | debug

Starts a generic wx_object server and invokes Mod:init(Args) in the new process.

start_link(Mod, Args, Options) -> wxWindow() (see module wxWindow)

Types:

Mod = atom()
Args = term()
Options = [{timeout, Timeout} | {debug, [Flag]}]
Flag = trace | log | {logfile, File} | statistics | debug

Starts a generic wx_object server and invokes Mod:init(Args) in the new process.

start_link(Name, Mod, Args, Options) -> wxWindow() (see module wxWindow)

Types:

Name = {local, atom()}
Mod = atom()
Args = term()
Options = [{timeout, Timeout} | {debug, [Flag]}]
Flag = trace | log | {logfile, File} | statistics | debug

Starts a generic wx_object server and invokes Mod:init(Args) in the new process.

call(Ref::wxObject() | atom() | pid(), Request::term()) -> term()

Make a call to a wx_object server. The call waits until it gets a result. Invokes handle_call(Request, From, State) in the server

call(Ref::wxObject() | atom() | pid(), Request::term(), Timeout::integer()) -> term()

Make a call to a wx_object server with a timeout. Invokes handle_call(Request, From, State) in server

cast(Ref::wxObject() | atom() | pid(), Request::term()) -> ok

Make a cast to a wx_object server. Invokes handle_cast(Request, State) in the server

`get_pid(Ref::wxObject()) -> pid()`

Get the pid of the object handle.

`reply(From::tuple(), Reply::term()) -> pid()`

Get the pid of the object handle.

wxAcceleratorEntry

Erlang module

See external documentation: **wxAcceleratorEntry**.

DATA TYPES

`wxAcceleratorEntry()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxAcceleratorEntry()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxAcceleratorEntry()`

Types:

```
Option = {flags, integer()} | {keyCode, integer()} | {cmd, integer()} |
{item, wxMenuItem() (see module wxMenuItem)}
```

See **external documentation**.
Also:
new(Entry) -> wxAcceleratorEntry() when
Entry::wxAcceleratorEntry().

`getCommand(This) -> integer()`

Types:

```
This = wxAcceleratorEntry()
```

See **external documentation**.

`getFlags(This) -> integer()`

Types:

```
This = wxAcceleratorEntry()
```

See **external documentation**.

`getKeyCode(This) -> integer()`

Types:

```
This = wxAcceleratorEntry()
```

See **external documentation**.

`set(This, Flags, KeyCode, Cmd) -> ok`

Types:

```
This = wxAcceleratorEntry()
Flags = integer()
```

```
KeyCode = integer()
Cmd = integer()
```

Equivalent to *set(This, Flags, KeyCode, Cmd, [])*.

`set(This, Flags, KeyCode, Cmd, Option::[Option]) -> ok`

Types:

```
This = wxAcceleratorEntry()
Flags = integer()
KeyCode = integer()
Cmd = integer()
Option = {item, wxMenuItem() (see module wxMenuItem)}
```

See **external documentation**.

`destroy(This::wxAcceleratorEntry()) -> ok`

Destroys this object, do not use object again

wxAcceleratorTable

Erlang module

See external documentation: **wxAcceleratorTable**.

DATA TYPES

`wxAcceleratorTable()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxAcceleratorTable()`

See **external documentation**.

`new(N, Entries) -> wxAcceleratorTable()`

Types:

```
N = integer()
Entries = [wxAcceleratorEntry() (see module wxAcceleratorEntry)]
```

See **external documentation**.

`ok(This) -> boolean()`

Types:

```
This = wxAcceleratorTable()
```

See **external documentation**.

`destroy(This::wxAcceleratorTable()) -> ok`

Destroys this object, do not use object again

wxArtProvider

Erlang module

See external documentation: **wxArtProvider**.

DATA TYPES

`wxArtProvider()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getBitmap(Id) -> wxBitmap() (see module wxBitmap)`

Types:

`Id = chardata() (see module unicode)`

Equivalent to *getBitmap(Id, [])*.

`getBitmap(Id, Option::[Option]) -> wxBitmap() (see module wxBitmap)`

Types:

`Id = chardata() (see module unicode)`

`Option = {client, chardata() (see module unicode)} | {size, {W::integer(), H::integer()}}`

See **external documentation**.

`getIcon(Id) -> wxIcon() (see module wxIcon)`

Types:

`Id = chardata() (see module unicode)`

Equivalent to *getIcon(Id, [])*.

`getIcon(Id, Option::[Option]) -> wxIcon() (see module wxIcon)`

Types:

`Id = chardata() (see module unicode)`

`Option = {client, chardata() (see module unicode)} | {size, {W::integer(), H::integer()}}`

See **external documentation**.

wxAuiDockArt

Erlang module

See external documentation: **wxAuiDockArt**.

DATA TYPES

`wxAuiDockArt()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxAuiManager

Erlang module

See external documentation: **wxAuiManager**.

This class is derived (and can use functions) from:
wxEvtHandler

DATA TYPES

`wxAuiManager()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxAuiManager()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxAuiManager()`

Types:

`Option = {managed_wnd, wxWindow() (see module wxWindow)} | {flags, integer()}`

See **external documentation**.

`addPane(This, Window) -> boolean()`

Types:

`This = wxAuiManager()`

`Window = wxWindow() (see module wxWindow)`

Equivalent to *addPane(This, Window, [])*.

`addPane(This, Window, Option::[Option]) -> boolean()`

Types:

`This = wxAuiManager()`

`Window = wxWindow() (see module wxWindow)`

`Option = {direction, integer()} | {caption, chardata() (see module unicode)}`

See **external documentation**.
Also:
addPane(This, Window, Pane_info) -> boolean() when
This::wxAuiManager(), Window::wxWindow:wxWindow(), Pane_info::wxAuiPaneInfo:wxAuiPaneInfo().

`addPane(This, Window, Pane_info, Drop_pos) -> boolean()`

Types:

`This = wxAuiManager()`

```
Window = wxWindow() (see module wxWindow)
Pane_info = wxAuiPaneInfo() (see module wxAuiPaneInfo)
Drop_pos = {X::integer(), Y::integer()}
```

See **external documentation**.

detachPane(This, Window) -> boolean()

Types:

```
This = wxAuiManager()
Window = wxWindow() (see module wxWindow)
```

See **external documentation**.

getAllPanes(This) -> wx_object() (see module wx)

Types:

```
This = wxAuiManager()
```

See **external documentation**.

getArtProvider(This) -> wxAuiDockArt() (see module wxAuiDockArt)

Types:

```
This = wxAuiManager()
```

See **external documentation**.

getDockSizeConstraint(This) -> {Width_pct::number(), Height_pct::number()}

Types:

```
This = wxAuiManager()
```

See **external documentation**.

getFlags(This) -> integer()

Types:

```
This = wxAuiManager()
```

See **external documentation**.

getManagedWindow(This) -> wxWindow() (see module wxWindow)

Types:

```
This = wxAuiManager()
```

See **external documentation**.

getManager(Window) -> wxAuiManager()

Types:

```
Window = wxWindow() (see module wxWindow)
```

See **external documentation**.

getPane(This, Name) -> wxAuiPaneInfo() (see module wxAuiPaneInfo)

Types:

```
This = wxAuiManager()
Name = chardata() (see module unicode)
```

See **external documentation**.
Also:
getPane(This, Window) -> wxAuiPaneInfo:wxAuiPaneInfo() when
This::wxAuiManager(), Window::wxWindow:wxWindow().

`hideHint(This) -> ok`

Types:

```
This = wxAuiManager()
```

See **external documentation**.

`insertPane(This, Window, Insert_location) -> boolean()`

Types:

```
This = wxAuiManager()
Window = wxWindow() (see module wxWindow)
Insert_location = wxAuiPaneInfo() (see module wxAuiPaneInfo)
```

Equivalent to *insertPane(This, Window, Insert_location, [])*.

`insertPane(This, Window, Insert_location, Option::[Option]) -> boolean()`

Types:

```
This = wxAuiManager()
Window = wxWindow() (see module wxWindow)
Insert_location = wxAuiPaneInfo() (see module wxAuiPaneInfo)
Option = {insert_level, integer()}
```

See **external documentation**.

`loadPaneInfo(This, Pane_part, Pane) -> ok`

Types:

```
This = wxAuiManager()
Pane_part = chardata() (see module unicode)
Pane = wxAuiPaneInfo() (see module wxAuiPaneInfo)
```

See **external documentation**.

`loadPerspective(This, Perspective) -> boolean()`

Types:

```
This = wxAuiManager()
Perspective = chardata() (see module unicode)
```

Equivalent to *loadPerspective(This, Perspective, [])*.

`loadPerspective(This, Perspective, Option::[Option]) -> boolean()`

Types:

```
This = wxAuiManager()
```

```
Perspective = chardata() (see module unicode)
Option = {update, boolean()}
```

See **external documentation**.

savePaneInfo(This, Pane) -> charlist() (see module unicode)

Types:

```
This = wxAuiManager()
Pane = wxAuiPaneInfo() (see module wxAuiPaneInfo)
```

See **external documentation**.

savePerspective(This) -> charlist() (see module unicode)

Types:

```
This = wxAuiManager()
```

See **external documentation**.

setArtProvider(This, Art_provider) -> ok

Types:

```
This = wxAuiManager()
Art_provider = wxAuiDockArt() (see module wxAuiDockArt)
```

See **external documentation**.

setDockSizeConstraint(This, Width_pct, Height_pct) -> ok

Types:

```
This = wxAuiManager()
Width_pct = number()
Height_pct = number()
```

See **external documentation**.

setFlags(This, Flags) -> ok

Types:

```
This = wxAuiManager()
Flags = integer()
```

See **external documentation**.

setManagedWindow(This, Managed_wnd) -> ok

Types:

```
This = wxAuiManager()
Managed_wnd = wxWindow() (see module wxWindow)
```

See **external documentation**.

showHint(This, Rect) -> ok

Types:

```
This = wxAuiManager()
```

```
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

See **external documentation**.

unInit(This) -> ok

Types:

```
This = wxAuiManager()
```

See **external documentation**.

update(This) -> ok

Types:

```
This = wxAuiManager()
```

See **external documentation**.

destroy(This::wxAuiManager()) -> ok

Destroys this object, do not use object again

wxAuiManagerEvent

Erlang module

See external documentation: **wxAuiManagerEvent**.

Use *wxEvtHandler:connect/3* with EventType:

aui_pane_button, *aui_pane_close*, *aui_pane_maximize*, *aui_pane_restore*, *aui_render*, *aui_find_manager*

See also the message variant *#wxAuiManager{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxAuiManagerEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`setManager(This, Mgr) -> ok`

Types:

`This = wxAuiManagerEvent()`
`Mgr = wxAuiManager() (see module wxAuiManager)`

See **external documentation**.

`getManager(This) -> wxAuiManager() (see module wxAuiManager)`

Types:

`This = wxAuiManagerEvent()`

See **external documentation**.

`setPane(This, P) -> ok`

Types:

`This = wxAuiManagerEvent()`
`P = wxAuiPaneInfo() (see module wxAuiPaneInfo)`

See **external documentation**.

`getPane(This) -> wxAuiPaneInfo() (see module wxAuiPaneInfo)`

Types:

`This = wxAuiManagerEvent()`

See **external documentation**.

`setButton(This, B) -> ok`

Types:

`This = wxAuiManagerEvent()`

```
B = integer()
```

See **external documentation**.

getButton(This) -> integer()

Types:

```
This = wxAuiManagerEvent()
```

See **external documentation**.

setDC(This, Pdc) -> ok

Types:

```
This = wxAuiManagerEvent()
Pdc = wxDC() (see module wxDC)
```

See **external documentation**.

getDC(This) -> wxDC() (see module wxDC)

Types:

```
This = wxAuiManagerEvent()
```

See **external documentation**.

veto(This) -> ok

Types:

```
This = wxAuiManagerEvent()
```

Equivalent to *veto(This, [])*.

veto(This, Option::[Option]) -> ok

Types:

```
This = wxAuiManagerEvent()
Option = {veto, boolean()}
```

See **external documentation**.

getVeto(This) -> boolean()

Types:

```
This = wxAuiManagerEvent()
```

See **external documentation**.

setCanVeto(This, Can_veto) -> ok

Types:

```
This = wxAuiManagerEvent()
Can_veto = boolean()
```

See **external documentation**.

canVeto(This) -> boolean()

Types:

```
This = wxAuiManagerEvent()
```

See **external documentation**.

wxAuiNotebook

Erlang module

See external documentation: **wxAuiNotebook**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxAuiNotebook()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxAuiNotebook()`

See **external documentation**.

`new(Parent) -> wxAuiNotebook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxAuiNotebook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`addPage(This, Page, Caption) -> boolean()`

Types:

```
This = wxAuiNotebook()
Page = wxWindow() (see module wxWindow)
Caption = chardata() (see module unicode)
```

Equivalent to *addPage(This, Page, Caption, [])*.

`addPage(This, Page, Caption, Option::[Option]) -> boolean()`

Types:

```
This = wxAuiNotebook()
Page = wxWindow() (see module wxWindow)
Caption = chardata() (see module unicode)
```

```
Option = {select, boolean()} | {bitmap, wxBitmap() (see module wxBitmap)}
```

See **external documentation**.

create(This, Parent) -> boolean()

Types:

```
This = wxAuiNotebook()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

create(This, Parent, Option::[Option]) -> boolean()

Types:

```
This = wxAuiNotebook()
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

deletePage(This, Page) -> boolean()

Types:

```
This = wxAuiNotebook()
Page = integer()
```

See **external documentation**.

getArtProvider(This) -> wxAuiTabArt() (see module wxAuiTabArt)

Types:

```
This = wxAuiNotebook()
```

See **external documentation**.

getPage(This, Page_idx) -> wxWindow() (see module wxWindow)

Types:

```
This = wxAuiNotebook()
Page_idx = integer()
```

See **external documentation**.

getPageBitmap(This, Page_idx) -> wxBitmap() (see module wxBitmap)

Types:

```
This = wxAuiNotebook()
Page_idx = integer()
```

See **external documentation**.

getPageCount(This) -> integer()

Types:

```
This = wxAuiNotebook()
```

See **external documentation**.

getPageIndex(This, Page_wnd) -> integer()

Types:

This = wxAuiNotebook()
Page_wnd = wxWindow() (see module wxWindow)

See **external documentation**.

getPageText(This, Page_idx) -> charlist() (see module unicode)

Types:

This = wxAuiNotebook()
Page_idx = integer()

See **external documentation**.

getSelection(This) -> integer()

Types:

This = wxAuiNotebook()

See **external documentation**.

insertPage(This, Page_idx, Page, Caption) -> boolean()

Types:

This = wxAuiNotebook()
Page_idx = integer()
Page = wxWindow() (see module wxWindow)
Caption = chardata() (see module unicode)

Equivalent to *insertPage(This, Page_idx, Page, Caption, [])*.

insertPage(This, Page_idx, Page, Caption, Option::[Option]) -> boolean()

Types:

This = wxAuiNotebook()
Page_idx = integer()
Page = wxWindow() (see module wxWindow)
Caption = chardata() (see module unicode)
Option = {select, boolean()} | {bitmap, wxBitmap() (see module wxBitmap)}

See **external documentation**.

removePage(This, Page) -> boolean()

Types:

This = wxAuiNotebook()
Page = integer()

See **external documentation**.

setArtProvider(This, Art) -> ok

Types:

```
This = wxAuiNotebook()
Art = wxAuiTabArt() (see module wxAuiTabArt)
```

See **external documentation**.

setFont(This, Font) -> boolean()

Types:

```
This = wxAuiNotebook()
Font = wxFont() (see module wxFont)
```

See **external documentation**.

setPageBitmap(This, Page, Bitmap) -> boolean()

Types:

```
This = wxAuiNotebook()
Page = integer()
Bitmap = wxBitmap() (see module wxBitmap)
```

See **external documentation**.

setPageText(This, Page, Text) -> boolean()

Types:

```
This = wxAuiNotebook()
Page = integer()
Text = chardata() (see module unicode)
```

See **external documentation**.

setSelection(This, New_page) -> integer()

Types:

```
This = wxAuiNotebook()
New_page = integer()
```

See **external documentation**.

setTabCtrlHeight(This, Height) -> ok

Types:

```
This = wxAuiNotebook()
Height = integer()
```

See **external documentation**.

setUniformBitmapSize(This, Size) -> ok

Types:

```
This = wxAuiNotebook()
Size = {W::integer(), H::integer()}
```

See **external documentation**.

destroy(This::wxAuiNotebook()) -> ok

Destroys this object, do not use object again

wxAuiNotebookEvent

Erlang module

See external documentation: **wxAuiNotebookEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_auinotebook_page_close, *command_auinotebook_page_changed*, *command_auinotebook_page_changing*, *command_auinotebook_button*, *command_auinotebook_begin_drag*, *command_auinotebook_end_drag*, *command_auinotebook_drag_motion*, *command_auinotebook_allow_dnd*, *command_auinotebook_tab_middle_down*, *command_auinotebook_tab_middle_up*, *command_auinotebook_tab_right_down*, *command_auinotebook_tab_right_up*, *command_auinotebook_page_closed*, *command_auinotebook_drag_done*, *command_auinotebook_bg_dclick*

See also the message variant *#wxAuiNotebook{}* event record type.

This class is derived (and can use functions) from:
wxNotifyEvent
wxCommandEvent
wxEvent

DATA TYPES

`wxAuiNotebookEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`setSelection(This, S) -> ok`

Types:

`This = wxAuiNotebookEvent()`
`S = integer()`

See **external documentation**.

`getSelection(This) -> integer()`

Types:

`This = wxAuiNotebookEvent()`

See **external documentation**.

`setOldSelection(This, S) -> ok`

Types:

`This = wxAuiNotebookEvent()`
`S = integer()`

See **external documentation**.

`getOldSelection(This) -> integer()`

Types:

```
This = wxAuiNotebookEvent()
```

See **external documentation**.

setDragSource(This, S) -> ok

Types:

```
This = wxAuiNotebookEvent()
S = wxAuiNotebook() (see module wxAuiNotebook)
```

See **external documentation**.

getDragSource(This) -> wxAuiNotebook() (see module wxAuiNotebook)

Types:

```
This = wxAuiNotebookEvent()
```

See **external documentation**.

wxAuiPaneInfo

Erlang module

See external documentation: **wxAuiPaneInfo**.

DATA TYPES

`wxAuiPaneInfo()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxAuiPaneInfo()`

See **external documentation**.

`new(C) -> wxAuiPaneInfo()`

Types:

```
C = wxAuiPaneInfo()
```

See **external documentation**.

`bestSize(This, Size) -> wxAuiPaneInfo()`

Types:

```
This = wxAuiPaneInfo()
Size = {W::integer(), H::integer()}
```

See **external documentation**.

`bestSize(This, X, Y) -> wxAuiPaneInfo()`

Types:

```
This = wxAuiPaneInfo()
X = integer()
Y = integer()
```

See **external documentation**.

`bottom(This) -> wxAuiPaneInfo()`

Types:

```
This = wxAuiPaneInfo()
```

See **external documentation**.

`bottomDockable(This) -> wxAuiPaneInfo()`

Types:

```
This = wxAuiPaneInfo()
```

Equivalent to *bottomDockable(This, [])*.

bottomDockable(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {b, boolean()}

See **external documentation**.

caption(This, C) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

C = chardata() (see module unicode)

See **external documentation**.

captionVisible(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *captionVisible(This, [])*.

captionVisible(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {visible, boolean()}

See **external documentation**.

centre(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

centrePane(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

closeButton(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *closeButton(This, [])*.

closeButton(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {visible, boolean()}

See **external documentation**.

defaultPane(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

destroyOnClose(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *destroyOnClose(This, [])*.

destroyOnClose(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {b, boolean()}

See **external documentation**.

direction(This, Direction) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Direction = integer()

See **external documentation**.

dock(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

dockable(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *dockable(This, [])*.

dockable(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {b, boolean()}

See **external documentation**.

fixed(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

float(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

floatable(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *floatable(This, [])*.

floatable(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {b, boolean()}

See **external documentation**.

floatingPosition(This, Pos) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Pos = {X::integer(), Y::integer()}

See **external documentation**.

floatingPosition(This, X, Y) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

X = integer()

Y = integer()

See **external documentation**.

floatingSize(This, Size) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Size = {W::integer(), H::integer()}

See **external documentation**.

floatingSize(This, X, Y) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

X = integer()

Y = integer()

See **external documentation**.

gripper(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *gripper(This, [])*.

gripper(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {visible, boolean()}

See **external documentation**.

gripperTop(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *gripperTop(This, [])*.

gripperTop(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {attop, boolean()}

See **external documentation**.

hasBorder(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

hasCaption(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

hasCloseButton(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

hasFlag(This, Flag) -> boolean()

Types:

This = wxAuiPaneInfo()

Flag = integer()

See **external documentation**.

hasGripper(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

hasGripperTop(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

hasMaximizeButton(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

hasMinimizeButton(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

hasPinButton(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

hide(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isBottomDockable(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isDocked(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isFixed(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isFloatable(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isFloating(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isLeftDockable(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isMovable(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isResizable(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isRightDockable(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isShown(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isToolbar(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

isTopDockable(This) -> boolean()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

layer(This, Layer) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Layer = integer()

See **external documentation**.

left(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

leftDockable(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *leftDockable(This, [])*.

leftDockable(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {b, boolean()}

See **external documentation**.

maxSize(This, Size) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Size = {W::integer(), H::integer()}

See **external documentation**.

maxSize(This, X, Y) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

X = integer()

Y = integer()

See **external documentation**.

maximizeButton(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *maximizeButton(This, [])*.

maximizeButton(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()
Option = {visible, boolean()}

See **external documentation**.

minSize(This, Size) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()
Size = {W::integer(), H::integer()}

See **external documentation**.

minSize(This, X, Y) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()
X = integer()
Y = integer()

See **external documentation**.

minimizeButton(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *minimizeButton(This, [])*.

minimizeButton(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()
Option = {visible, boolean()}

See **external documentation**.

movable(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *movable(This, [])*.

movable(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {b, boolean()}

See **external documentation**.

name(This, N) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

N = chardata() (see module unicode)

See **external documentation**.

paneBorder(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *paneBorder(This, [])*.

paneBorder(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {visible, boolean()}

See **external documentation**.

pinButton(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *pinButton(This, [])*.

pinButton(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {visible, boolean()}

See **external documentation**.

position(This, Pos) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Pos = integer()

See **external documentation**.

resizable(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *resizable(This, [])*.

resizable(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()
Option = {resizable, boolean()}

See **external documentation**.

right(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

rightDockable(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *rightDockable(This, [])*.

rightDockable(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()
Option = {b, boolean()}

See **external documentation**.

row(This, Row) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()
Row = integer()

See **external documentation**.

safeSet(This, Source) -> ok

Types:

This = wxAuiPaneInfo()
Source = wxAuiPaneInfo()

See **external documentation**.

setFlag(This, Flag, Option_state) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()
Flag = integer()
Option_state = boolean()

See **external documentation**.

show(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *show(This, [])*.

show(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {show, boolean()}

See **external documentation**.

toolbarPane(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

top(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

See **external documentation**.

topDockable(This) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Equivalent to *topDockable(This, [])*.

topDockable(This, Option::[Option]) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

Option = {b, boolean()}

See **external documentation**.

window(This, W) -> wxAuiPaneInfo()

Types:

This = wxAuiPaneInfo()

W = wxWindow() (see module wxWindow)

See **external documentation**.

destroy(This::wxAuiPaneInfo()) -> ok

Destroys this object, do not use object again

wxAuiTabArt

Erlang module

See external documentation: **wxAuiTabArt**.

DATA TYPES

`wxAuiTabArt()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxBitmap

Erlang module

See external documentation: **wxBitmap**.

DATA TYPES

`wxBitmap()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxBitmap()`

See **external documentation**.

`new(Filename) -> wxBitmap()`

Types:

`Filename = chardata() (see module unicode)`

See **external documentation**.
Also:
new(Image) -> wxBitmap() when
Image::wxImage:wxImage().

Type = ?wxBITMAP_TYPE_INVALID | ?wxBITMAP_TYPE_BMP | ?wxBITMAP_TYPE_BMP_RESOURCE | ?wxBITMAP_TYPE_RESOURCE | ?wxBITMAP_TYPE_ICO | ?wxBITMAP_TYPE_ICO_RESOURCE | ?wxBITMAP_TYPE_CUR | ?wxBITMAP_TYPE_CUR_RESOURCE | ?wxBITMAP_TYPE_XBM | ?wxBITMAP_TYPE_XBM_DATA | ?wxBITMAP_TYPE_XPM | ?wxBITMAP_TYPE_XPM_DATA | ?wxBITMAP_TYPE_TIF | ?wxBITMAP_TYPE_TIF_RESOURCE | ?wxBITMAP_TYPE_GIF | ?wxBITMAP_TYPE_GIF_RESOURCE | ?wxBITMAP_TYPE_PNG | ?wxBITMAP_TYPE_PNG_RESOURCE | ?wxBITMAP_TYPE_JPEG | ?wxBITMAP_TYPE_JPEG_RESOURCE | ?wxBITMAP_TYPE_PNM | ?wxBITMAP_TYPE_PNM_RESOURCE | ?wxBITMAP_TYPE_PCX | ?wxBITMAP_TYPE_PCX_RESOURCE | ?wxBITMAP_TYPE_PICT | ?wxBITMAP_TYPE_PICT_RESOURCE | ?wxBITMAP_TYPE_ICON | ?wxBITMAP_TYPE_ICON_RESOURCE | ?wxBITMAP_TYPE_ANI | ?wxBITMAP_TYPE_IFF | ?wxBITMAP_TYPE_TGA | ?wxBITMAP_TYPE_MACCURSOR | ?wxBITMAP_TYPE_MACCURSOR_RESOURCE | ?wxBITMAP_TYPE_ANY

`new(Width, Height) -> wxBitmap()`

Types:

`Width = integer()`

`Height = integer()`

See **external documentation**.
Also:
new(Filename, [Option]) -> wxBitmap() when
Filename::unicode:chardata(),
Option :: {type, wx:wx_enum()};
(Image, [Option]) -> wxBitmap() when

Image::wxImage:wxImage(),
Option :: {depth, integer()}.

Type = ?wxBITMAP_TYPE_INVALID | ?wxBITMAP_TYPE_BMP | ?wxBITMAP_TYPE_BMP_RESOURCE | ?wxBITMAP_TYPE_RESOURCE | ?wxBITMAP_TYPE_ICO | ?wxBITMAP_TYPE_ICO_RESOURCE | ?wxBITMAP_TYPE_CUR | ?wxBITMAP_TYPE_CUR_RESOURCE | ?wxBITMAP_TYPE_XBM | ?wxBITMAP_TYPE_XBM_DATA | ?wxBITMAP_TYPE_XPM | ?wxBITMAP_TYPE_XPM_DATA | ?wxBITMAP_TYPE_TIF | ?wxBITMAP_TYPE_TIF_RESOURCE | ?wxBITMAP_TYPE_GIF | ?wxBITMAP_TYPE_GIF_RESOURCE | ?wxBITMAP_TYPE_PNG | ?wxBITMAP_TYPE_PNG_RESOURCE | ?wxBITMAP_TYPE_JPEG | ?wxBITMAP_TYPE_JPEG_RESOURCE | ?wxBITMAP_TYPE_PNM | ?wxBITMAP_TYPE_PNM_RESOURCE | ?wxBITMAP_TYPE_PCX | ?wxBITMAP_TYPE_PCX_RESOURCE | ?wxBITMAP_TYPE_PICT | ?wxBITMAP_TYPE_PICT_RESOURCE | ?wxBITMAP_TYPE_ICON | ?wxBITMAP_TYPE_ICON_RESOURCE | ?wxBITMAP_TYPE_ANI | ?wxBITMAP_TYPE_IFF | ?wxBITMAP_TYPE_TGA | ?wxBITMAP_TYPE_MACCURSOR | ?wxBITMAP_TYPE_MACCURSOR_RESOURCE | ?wxBITMAP_TYPE_ANY

`new(Bits, Width, Height) -> wxBitmap()`

Types:

```
Bits = binary()
Width = integer()
Height = integer()
```

See **external documentation**.
Also:
new(Width, Height, [Option]) -> wxBitmap() when
Width::integer(), Height::integer(),
Option :: {depth, integer()}.

`new(Bits, Width, Height, Option::[Option]) -> wxBitmap()`

Types:

```
Bits = binary()
Width = integer()
Height = integer()
Option = {depth, integer()}
```

See **external documentation**.

`convertToImage(This) -> wxImage() (see module wxImage)`

Types:

```
This = wxBitmap()
```

See **external documentation**.

`copyFromIcon(This, Icon) -> boolean()`

Types:

```
This = wxBitmap()
Icon = wxIcon() (see module wxIcon)
```

See **external documentation**.

create(This, Width, Height) -> boolean()

Types:

This = wxBitmap()
Width = integer()
Height = integer()

Equivalent to *create(This, Width, Height, [])*.

create(This, Width, Height, Option::[Option]) -> boolean()

Types:

This = wxBitmap()
Width = integer()
Height = integer()
Option = {depth, integer()}

See **external documentation**.

getDepth(This) -> integer()

Types:

This = wxBitmap()

See **external documentation**.

getHeight(This) -> integer()

Types:

This = wxBitmap()

See **external documentation**.

getPalette(This) -> wxPalette() (see module wxPalette)

Types:

This = wxBitmap()

See **external documentation**.

getMask(This) -> wxMask() (see module wxMask)

Types:

This = wxBitmap()

See **external documentation**.

getWidth(This) -> integer()

Types:

This = wxBitmap()

See **external documentation**.

getSubBitmap(This, Rect) -> wxBitmap()

Types:

This = wxBitmap()

```
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

See **external documentation**.

loadFile(This, Name) -> boolean()

Types:

```
This = wxBitmap()
Name = chardata() (see module unicode)
```

Equivalent to *loadFile(This, Name, [])*.

loadFile(This, Name, Option::[Option]) -> boolean()

Types:

```
This = wxBitmap()
Name = chardata() (see module unicode)
Option = {type, wx_enum() (see module wx)}
```

See **external documentation**.
Type = ?wxBITMAP_TYPE_INVALID | ?wxBITMAP_TYPE_BMP | ?wxBITMAP_TYPE_BMP_RESOURCE | ?wxBITMAP_TYPE_RESOURCE | ?wxBITMAP_TYPE_ICO | ?wxBITMAP_TYPE_ICO_RESOURCE | ?wxBITMAP_TYPE_CUR | ?wxBITMAP_TYPE_CUR_RESOURCE | ?wxBITMAP_TYPE_XBM | ?wxBITMAP_TYPE_XBM_DATA | ?wxBITMAP_TYPE_XPM | ?wxBITMAP_TYPE_XPM_DATA | ?wxBITMAP_TYPE_TIF | ?wxBITMAP_TYPE_TIF_RESOURCE | ?wxBITMAP_TYPE_GIF | ?wxBITMAP_TYPE_GIF_RESOURCE | ?wxBITMAP_TYPE_PNG | ?wxBITMAP_TYPE_PNG_RESOURCE | ?wxBITMAP_TYPE_JPEG | ?wxBITMAP_TYPE_JPEG_RESOURCE | ?wxBITMAP_TYPE_PNM | ?wxBITMAP_TYPE_PNM_RESOURCE | ?wxBITMAP_TYPE_PCX | ?wxBITMAP_TYPE_PCX_RESOURCE | ?wxBITMAP_TYPE_PICT | ?wxBITMAP_TYPE_PICT_RESOURCE | ?wxBITMAP_TYPE_ICON | ?wxBITMAP_TYPE_ICON_RESOURCE | ?wxBITMAP_TYPE_ANI | ?wxBITMAP_TYPE_IFF | ?wxBITMAP_TYPE_TGA | ?wxBITMAP_TYPE_MACCURSOR | ?wxBITMAP_TYPE_MACCURSOR_RESOURCE | ?wxBITMAP_TYPE_ANY

ok(This) -> boolean()

Types:

```
This = wxBitmap()
```

See **external documentation**.

saveFile(This, Name, Type) -> boolean()

Types:

```
This = wxBitmap()
Name = chardata() (see module unicode)
Type = wx_enum() (see module wx)
```

Equivalent to *saveFile(This, Name, Type, [])*.

saveFile(This, Name, Type, Option::[Option]) -> boolean()

Types:

```
This = wxBitmap()
Name = chardata() (see module unicode)
Type = wx_enum() (see module wx)
```

Option = {palette, wxPalette() (see module wxPalette)}

See **external documentation**.
Type = ?wxBITMAP_TYPE_INVALID | ?wxBITMAP_TYPE_BMP | ?wxBITMAP_TYPE_BMP_RESOURCE | ?wxBITMAP_TYPE_RESOURCE | ?wxBITMAP_TYPE_ICO | ?wxBITMAP_TYPE_ICO_RESOURCE | ?wxBITMAP_TYPE_CUR | ?wxBITMAP_TYPE_CUR_RESOURCE | ?wxBITMAP_TYPE_XBM | ?wxBITMAP_TYPE_XBM_DATA | ?wxBITMAP_TYPE_XPM | ?wxBITMAP_TYPE_XPM_DATA | ?wxBITMAP_TYPE_TIF | ?wxBITMAP_TYPE_TIF_RESOURCE | ?wxBITMAP_TYPE_GIF | ?wxBITMAP_TYPE_GIF_RESOURCE | ?wxBITMAP_TYPE_PNG | ?wxBITMAP_TYPE_PNG_RESOURCE | ?wxBITMAP_TYPE_JPEG | ?wxBITMAP_TYPE_JPEG_RESOURCE | ?wxBITMAP_TYPE_PNM | ?wxBITMAP_TYPE_PNM_RESOURCE | ?wxBITMAP_TYPE_PCX | ?wxBITMAP_TYPE_PCX_RESOURCE | ?wxBITMAP_TYPE_PICT | ?wxBITMAP_TYPE_PICT_RESOURCE | ?wxBITMAP_TYPE_ICON | ?wxBITMAP_TYPE_ICON_RESOURCE | ?wxBITMAP_TYPE_ANI | ?wxBITMAP_TYPE_IFF | ?wxBITMAP_TYPE_TGA | ?wxBITMAP_TYPE_MACCURSOR | ?wxBITMAP_TYPE_MACCURSOR_RESOURCE | ?wxBITMAP_TYPE_ANY

setDepth(This, Depth) -> ok

Types:

This = wxBitmap()
Depth = integer()

See **external documentation**.

setHeight(This, Height) -> ok

Types:

This = wxBitmap()
Height = integer()

See **external documentation**.

setMask(This, Mask) -> ok

Types:

This = wxBitmap()
Mask = wxMask() (see module wxMask)

See **external documentation**.

setPalette(This, Palette) -> ok

Types:

This = wxBitmap()
Palette = wxPalette() (see module wxPalette)

See **external documentation**.

setWidth(This, Width) -> ok

Types:

This = wxBitmap()
Width = integer()

See **external documentation**.

destroy(This::wxBitmap()) -> ok

Destroys this object, do not use object again

wxBitmapButton

Erlang module

See external documentation: **wxBitmapButton**.

This class is derived (and can use functions) from:
wxButton
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxBitmapButton()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxBitmapButton()`

See **external documentation**.

`new(Parent, Id, Bitmap) -> wxBitmapButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Bitmap = wxBitmap() (see module wxBitmap)
```

Equivalent to *new(Parent, Id, Bitmap, [])*.

`new(Parent, Id, Bitmap, Option::[Option]) -> wxBitmapButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Bitmap = wxBitmap() (see module wxBitmap)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

`create(This, Parent, Id, Bitmap) -> boolean()`

Types:

```
This = wxBitmapButton()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Bitmap = wxBitmap() (see module wxBitmap)
```

Equivalent to *create(This, Parent, Id, Bitmap, [])*.

create(This, Parent, Id, Bitmap, Option::[Option]) -> boolean()

Types:

This = wxBitmapButton()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Bitmap = wxBitmap() (see module wxBitmap)

Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {validator, wx_object() (see module wx)}

See **external documentation**.

getBitmapDisabled(This) -> wxBitmap() (see module wxBitmap)

Types:

This = wxBitmapButton()

See **external documentation**.

getBitmapFocus(This) -> wxBitmap() (see module wxBitmap)

Types:

This = wxBitmapButton()

See **external documentation**.

getBitmapLabel(This) -> wxBitmap() (see module wxBitmap)

Types:

This = wxBitmapButton()

See **external documentation**.

getBitmapSelected(This) -> wxBitmap() (see module wxBitmap)

Types:

This = wxBitmapButton()

See **external documentation**.

setBitmapDisabled(This, Disabled) -> ok

Types:

This = wxBitmapButton()

Disabled = wxBitmap() (see module wxBitmap)

See **external documentation**.

setBitmapFocus(This, Focus) -> ok

Types:

This = wxBitmapButton()

Focus = wxBitmap() (see module wxBitmap)

See **external documentation**.

setBitmapLabel(This, Bitmap) -> ok

Types:

```
This = wxBitmapButton()
Bitmap = wxBitmap() (see module wxBitmap)
```

See **external documentation**.

setBitmapSelected(This, Sel) -> ok

Types:

```
This = wxBitmapButton()
Sel = wxBitmap() (see module wxBitmap)
```

See **external documentation**.

destroy(This::wxBitmapButton()) -> ok

Destroys this object, do not use object again

wxBitmapDataObject

Erlang module

See external documentation: **wxBitmapDataObject**.

This class is derived (and can use functions) from:
wxDataObject

DATA TYPES

`wxBitmapDataObject()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxBitmapDataObject()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxBitmapDataObject()`

Types:

`Option = {bitmap, wxBitmap() (see module wxBitmap)}`

See **external documentation**.
Also:
new(Bitmap) -> wxBitmapDataObject() when
Bitmap::wxBitmap:wxBitmap().

`getBitmap(This) -> wxBitmap() (see module wxBitmap)`

Types:

`This = wxBitmapDataObject()`

See **external documentation**.

`setBitmap(This, Bitmap) -> ok`

Types:

`This = wxBitmapDataObject()`
`Bitmap = wxBitmap() (see module wxBitmap)`

See **external documentation**.

`destroy(This::wxBitmapDataObject()) -> ok`

Destroys this object, do not use object again

wxBoxSizer

Erlang module

See external documentation: **wxBoxSizer**.

This class is derived (and can use functions) from:
wxSizer

DATA TYPES

`wxBoxSizer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Orient) -> wxBoxSizer()`

Types:

`Orient = integer()`

See **external documentation**.

`getOrientation(This) -> integer()`

Types:

`This = wxBoxSizer()`

See **external documentation**.

`destroy(This::wxBoxSizer()) -> ok`

Destroys this object, do not use object again

wxBrush

Erlang module

See external documentation: **wxBrush**.

DATA TYPES

`wxBrush()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxBrush()`

See **external documentation**.

`new(Colour) -> wxBrush()`

Types:

`Colour = wx_colour() (see module wx)`

See **external documentation**.
Also:
new(StippleBitmap) -> wxBrush() when
StippleBitmap::wxBitmap:wxBitmap().

`new(Colour, Option::[Option]) -> wxBrush()`

Types:

`Colour = wx_colour() (see module wx)`
`Option = {style, integer()}`

See **external documentation**.

`getColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxBrush()`

See **external documentation**.

`getStipple(This) -> wxBitmap() (see module wxBitmap)`

Types:

`This = wxBrush()`

See **external documentation**.

`getStyle(This) -> integer()`

Types:

`This = wxBrush()`

See **external documentation**.

isHatch(This) -> boolean()

Types:

This = wxBrush()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxBrush()

See **external documentation**.

setColour(This, Col) -> ok

Types:

This = wxBrush()
Col = wx_colour() (see module wx)

See **external documentation**.

setColour(This, R, G, B) -> ok

Types:

This = wxBrush()
R = integer()
G = integer()
B = integer()

See **external documentation**.

setStipple(This, Stipple) -> ok

Types:

This = wxBrush()
Stipple = wxBitmap() (see module wxBitmap)

See **external documentation**.

setStyle(This, Style) -> ok

Types:

This = wxBrush()
Style = integer()

See **external documentation**.

destroy(This::wxBrush()) -> ok

Destroys this object, do not use object again

wxBufferedDC

Erlang module

See external documentation: **wxBufferedDC**.

This class is derived (and can use functions) from:
wxMemoryDC
wxDC

DATA TYPES

`wxBufferedDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxBufferedDC()`

See **external documentation**.

`new(Dc) -> wxBufferedDC()`

Types:

`Dc = wxDC() (see module wxDC)`

Equivalent to *new(Dc, [])*.

`new(Dc, Area) -> wxBufferedDC()`

Types:

`Dc = wxDC() (see module wxDC)`
`Area = {W::integer(), H::integer()}`

See **external documentation**.
Also:
new(Dc, [Option]) -> wxBufferedDC() when
Dc::wxDC:wxDC(),
Option :: {buffer, wxBitmap:wxBitmap()}
| {style, integer()}.

`new(Dc, Area, Option::[Option]) -> wxBufferedDC()`

Types:

`Dc = wxDC() (see module wxDC)`
`Area = {W::integer(), H::integer()}`
`Option = {style, integer()}`

See **external documentation**.

`init(This, Dc) -> ok`

Types:

```
This = wxBufferedDC()
Dc = wxDC() (see module wxDC)
```

Equivalent to *init(This, Dc, [])*.

`init(This, Dc, Area) -> ok`

Types:

```
This = wxBufferedDC()
Dc = wxDC() (see module wxDC)
Area = {W::integer(), H::integer()}
```

See **external documentation**.
Also:
init(This, Dc, [Option]) -> ok when
This::wxBufferedDC(), Dc::wxDC:wxDC(),
Option :: {buffer, wxBitmap:wxBitmap()}
| {style, integer()}.

`init(This, Dc, Area, Option::[Option]) -> ok`

Types:

```
This = wxBufferedDC()
Dc = wxDC() (see module wxDC)
Area = {W::integer(), H::integer()}
Option = {style, integer()}
```

See **external documentation**.

`destroy(This::wxBufferedDC()) -> ok`

Destroys this object, do not use object again

wxBufferedPaintDC

Erlang module

See external documentation: **wxBufferedPaintDC**.

This class is derived (and can use functions) from:
wxBufferedDC
wxMemoryDC
wxDC

DATA TYPES

`wxBufferedPaintDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Window) -> wxBufferedPaintDC()`

Types:

`Window = wxWindow() (see module wxWindow)`

Equivalent to *new(Window, [])*.

`new(Window, Buffer) -> wxBufferedPaintDC()`

Types:

`Window = wxWindow() (see module wxWindow)`
`Buffer = wxBitmap() (see module wxBitmap)`

See **external documentation**.
Also:
new(Window, [Option]) -> wxBufferedPaintDC() when
Window::wxWindow:wxWindow(),
Option :: {style, integer()}.

`new(Window, Buffer, Option::[Option]) -> wxBufferedPaintDC()`

Types:

`Window = wxWindow() (see module wxWindow)`
`Buffer = wxBitmap() (see module wxBitmap)`
`Option = {style, integer()}`

See **external documentation**.

`destroy(This::wxBufferedPaintDC()) -> ok`

Destroys this object, do not use object again

wxButton

Erlang module

See external documentation: **wxButton**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxButton()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxButton()`

See **external documentation**.

`new(Parent, Id) -> wxButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {label, chardata() (see module unicode)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
| {validator, wx_object() (see module wx)}
```

See **external documentation**.

`create(This, Parent, Id) -> boolean()`

Types:

```
This = wxButton()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

`create(This, Parent, Id, Option::[Option]) -> boolean()`

Types:

```
This = wxButton()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {label, chardata() (see module unicode)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
| {validator, wx_object() (see module wx)}
```

See **external documentation**.

getDefaultSize() -> {W::integer(), H::integer()}

See **external documentation**.

setDefault(This) -> ok

Types:

```
This = wxButton()
```

See **external documentation**.

setLabel(This, Label) -> ok

Types:

```
This = wxButton()
Label = chardata() (see module unicode)
```

See **external documentation**.

destroy(This::wxButton()) -> ok

Destroys this object, do not use object again

wxCalendarCtrl

Erlang module

See external documentation: **wxCalendarCtrl**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxCalendarCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxCalendarCtrl()`

See **external documentation**.

`new(Parent, Id) -> wxCalendarCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxCalendarCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {date, wx_datetime() (see module wx)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent, Id) -> boolean()`

Types:

```
This = wxCalendarCtrl()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

`create(This, Parent, Id, Option::[Option]) -> boolean()`

Types:

```
This = wxCalendarCtrl()
```

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {date, wx_datetime() (see module wx)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

setDate(This, Date) -> boolean()

Types:

```
This = wxCalendarCtrl()
Date = wx_datetime() (see module wx)
```

See **external documentation**.

getDate(This) -> wx_datetime() (see module wx)

Types:

```
This = wxCalendarCtrl()
```

See **external documentation**.

enableYearChange(This) -> ok

Types:

```
This = wxCalendarCtrl()
```

Equivalent to *enableYearChange(This, [])*.

enableYearChange(This, Option::[Option]) -> ok

Types:

```
This = wxCalendarCtrl()
Option = {enable, boolean()}
```

See **external documentation**.

enableMonthChange(This) -> ok

Types:

```
This = wxCalendarCtrl()
```

Equivalent to *enableMonthChange(This, [])*.

enableMonthChange(This, Option::[Option]) -> ok

Types:

```
This = wxCalendarCtrl()
Option = {enable, boolean()}
```

See **external documentation**.

enableHolidayDisplay(This) -> ok

Types:

```
This = wxCalendarCtrl()
```

Equivalent to *enableHolidayDisplay(This, [])*.

enableHolidayDisplay(This, Option::[Option]) -> ok

Types:

This = wxCalendarCtrl()
Option = {display, boolean()}

See **external documentation**.

setHeaderColours(This, ColFg, ColBg) -> ok

Types:

This = wxCalendarCtrl()
ColFg = wx_colour() (see module wx)
ColBg = wx_colour() (see module wx)

See **external documentation**.

getHeaderColourFg(This) -> wx_colour4() (see module wx)

Types:

This = wxCalendarCtrl()

See **external documentation**.

getHeaderColourBg(This) -> wx_colour4() (see module wx)

Types:

This = wxCalendarCtrl()

See **external documentation**.

setHighlightColours(This, ColFg, ColBg) -> ok

Types:

This = wxCalendarCtrl()
ColFg = wx_colour() (see module wx)
ColBg = wx_colour() (see module wx)

See **external documentation**.

getHighlightColourFg(This) -> wx_colour4() (see module wx)

Types:

This = wxCalendarCtrl()

See **external documentation**.

getHighlightColourBg(This) -> wx_colour4() (see module wx)

Types:

This = wxCalendarCtrl()

See **external documentation**.

setHolidayColours(This, ColFg, ColBg) -> ok

Types:

This = wxCalendarCtrl()

```
ColFg = wx_colour() (see module wx)
ColBg = wx_colour() (see module wx)
```

See **external documentation**.

```
getHolidayColourFg(This) -> wx_colour4() (see module wx)
```

Types:

```
This = wxCalendarCtrl()
```

See **external documentation**.

```
getHolidayColourBg(This) -> wx_colour4() (see module wx)
```

Types:

```
This = wxCalendarCtrl()
```

See **external documentation**.

```
getAttr(This, Day) -> wxCalendarDateAttr() (see module wxCalendarDateAttr)
```

Types:

```
This = wxCalendarCtrl()
Day = integer()
```

See **external documentation**.

```
setAttr(This, Day, Attr) -> ok
```

Types:

```
This = wxCalendarCtrl()
Day = integer()
Attr = wxCalendarDateAttr() (see module wxCalendarDateAttr)
```

See **external documentation**.

```
setHoliday(This, Day) -> ok
```

Types:

```
This = wxCalendarCtrl()
Day = integer()
```

See **external documentation**.

```
resetAttr(This, Day) -> ok
```

Types:

```
This = wxCalendarCtrl()
Day = integer()
```

See **external documentation**.

```
hitTest(This, Pos) -> Result
```

Types:

```
Result = {Res::wx_enum() (see module wx), Date::wx_datetime() (see module
wx), Wd::wx_enum() (see module wx)}
```

```
This = wxCalendarCtrl()
Pos = {X::integer(), Y::integer()}
```

See **external documentation**.
Wd = ?wxDateTime_Sun | ?wxDateTime_Mon | ?wxDateTime_Tue | ?wxDateTime_Wed | ?wxDateTime_Thu | ?wxDateTime_Fri | ?wxDateTime_Sat | ?wxDateTime_Inv_WeekDay
Res = ?wxCAL_HITTEST_NOWHERE | ?wxCAL_HITTEST_HEADER | ?wxCAL_HITTEST_DAY | ?wxCAL_HITTEST_INCMONTH | ?wxCAL_HITTEST_DECMONTH | ?wxCAL_HITTEST_SURROUNDING_WEEK

`destroy(This::wxCalendarCtrl()) -> ok`

Destroys this object, do not use object again

wxCalendarDateAttr

Erlang module

See external documentation: **wxCalendarDateAttr**.

DATA TYPES

`wxCalendarDateAttr()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxCalendarDateAttr()`

See **external documentation**.

`new(Border) -> wxCalendarDateAttr()`

Types:

`Border = wx_enum() (see module wx)`

See **external documentation**.
Also:
new(ColText) -> wxCalendarDateAttr() when
ColText::wx:wx_colour().

Border = ?wxCAL_BORDER_NONE | ?wxCAL_BORDER_SQUARE | ?wxCAL_BORDER_ROUND

`new(Border, Option::[Option]) -> wxCalendarDateAttr()`

Types:

`Border = wx_enum() (see module wx)`
`Option = {colBorder, wx_colour() (see module wx)}`

See **external documentation**.
Also:
new(ColText, [Option]) -> wxCalendarDateAttr() when
ColText::wx:wx_colour(),
Option :: {colBack, wx:wx_colour()}
| {colBorder, wx:wx_colour()}
| {font, wxFont:wxFont()}
| {border, wx:wx_enum()}.

Border = ?wxCAL_BORDER_NONE | ?wxCAL_BORDER_SQUARE | ?wxCAL_BORDER_ROUND

`setTextColour(This, ColText) -> ok`

Types:

`This = wxCalendarDateAttr()`
`ColText = wx_colour() (see module wx)`

See **external documentation**.

setBackgroundColour(This, ColBack) -> ok

Types:

This = wxCalendarDateAttr()
ColBack = wx_colour() (see module wx)

See **external documentation**.

setBorderColour(This, Col) -> ok

Types:

This = wxCalendarDateAttr()
Col = wx_colour() (see module wx)

See **external documentation**.

setFont(This, Font) -> ok

Types:

This = wxCalendarDateAttr()
Font = wxFont() (see module wxFont)

See **external documentation**.

setBorder(This, Border) -> ok

Types:

This = wxCalendarDateAttr()
Border = wx_enum() (see module wx)

See **external documentation**.
Border = ?wxCAL_BORDER_NONE | ?wxCAL_BORDER_SQUARE | ?wxCAL_BORDER_ROUND

setHoliday(This, Holiday) -> ok

Types:

This = wxCalendarDateAttr()
Holiday = boolean()

See **external documentation**.

hasTextColour(This) -> boolean()

Types:

This = wxCalendarDateAttr()

See **external documentation**.

hasBackgroundColour(This) -> boolean()

Types:

This = wxCalendarDateAttr()

See **external documentation**.

hasBorderColour(This) -> boolean()

Types:

This = wxCalendarDateAttr()

See **external documentation**.

hasFont(This) -> boolean()

Types:

This = wxCalendarDateAttr()

See **external documentation**.

hasBorder(This) -> boolean()

Types:

This = wxCalendarDateAttr()

See **external documentation**.

isHoliday(This) -> boolean()

Types:

This = wxCalendarDateAttr()

See **external documentation**.

getTextColour(This) -> wx_colour4() (see module wx)

Types:

This = wxCalendarDateAttr()

See **external documentation**.

getBackgroundColour(This) -> wx_colour4() (see module wx)

Types:

This = wxCalendarDateAttr()

See **external documentation**.

getBorderColour(This) -> wx_colour4() (see module wx)

Types:

This = wxCalendarDateAttr()

See **external documentation**.

getFont(This) -> wxFont() (see module wxFont)

Types:

This = wxCalendarDateAttr()

See **external documentation**.

getBorder(This) -> wx_enum() (see module wx)

Types:

This = wxCalendarDateAttr()

See **external documentation**.

Res = ?wxCAL_BORDER_NONE | ?wxCAL_BORDER_SQUARE | ?wxCAL_BORDER_ROUND

destroy(This::wxCalendarDateAttr()) -> ok

Destroys this object, do not use object again

wxCalendarEvent

Erlang module

See external documentation: **wxCalendarEvent**.

Use *wxEvtHandler:connect/3* with EventType:

calendar_sel_changed, *calendar_day_changed*, *calendar_month_changed*, *calendar_year_changed*, *calendar_doubleclicked*, *calendar_weekday_clicked*

See also the message variant *#wxCalendar{}* event record type.

This class is derived (and can use functions) from:
wxDateEvent
wxCommandEvent
wxEvent

DATA TYPES

`wxCalendarEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getWeekDay(This) -> wx_enum() (see module wx)`

Types:

`This = wxCalendarEvent()`

See **external documentation**.
Res = ?wxDateTime_Sun | ?wxDateTime_Mon | ?wxDateTime_Tue | ?wxDateTime_Wed | ?wxDateTime_Thu | ?wxDateTime_Fri | ?wxDateTime_Sat | ?wxDateTime_Inv_WeekDay

wxCaret

Erlang module

See external documentation: **wxCaret**.

DATA TYPES

`wxCaret()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Window, Size) -> wxCaret()`

Types:

```
Window = wxWindow() (see module wxWindow)
Size = {W::integer(), H::integer()}
```

See **external documentation**.

`new(Window, Width, Height) -> wxCaret()`

Types:

```
Window = wxWindow() (see module wxWindow)
Width = integer()
Height = integer()
```

See **external documentation**.

`create(This, Window, Size) -> boolean()`

Types:

```
This = wxCaret()
Window = wxWindow() (see module wxWindow)
Size = {W::integer(), H::integer()}
```

See **external documentation**.

`create(This, Window, Width, Height) -> boolean()`

Types:

```
This = wxCaret()
Window = wxWindow() (see module wxWindow)
Width = integer()
Height = integer()
```

See **external documentation**.

`getBlinkTime() -> integer()`

See **external documentation**.

getPosition(This) -> {X::integer(), Y::integer()}

Types:

This = wxCaret()

See **external documentation**.

getSize(This) -> {W::integer(), H::integer()}

Types:

This = wxCaret()

See **external documentation**.

getWindow(This) -> wxWindow() (see module wxWindow)

Types:

This = wxCaret()

See **external documentation**.

hide(This) -> ok

Types:

This = wxCaret()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxCaret()

See **external documentation**.

isVisible(This) -> boolean()

Types:

This = wxCaret()

See **external documentation**.

move(This, Pt) -> ok

Types:

This = wxCaret()

Pt = {X::integer(), Y::integer()}

See **external documentation**.

move(This, X, Y) -> ok

Types:

This = wxCaret()

X = integer()

Y = integer()

See **external documentation**.

setBlinkTime(Milliseconds) -> ok

Types:

Milliseconds = integer()

See **external documentation**.

setSize(This, Size) -> ok

Types:

This = wxCaret()
Size = {W::integer(), H::integer()}

See **external documentation**.

setSize(This, Width, Height) -> ok

Types:

This = wxCaret()
Width = integer()
Height = integer()

See **external documentation**.

show(This) -> ok

Types:

This = wxCaret()

Equivalent to *show(This, [])*.

show(This, Option::[Option]) -> ok

Types:

This = wxCaret()
Option = {show, boolean()}

See **external documentation**.

destroy(This::wxCaret()) -> ok

Destroys this object, do not use object again

wxCheckBox

Erlang module

See external documentation: **wxCheckBox**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxCheckBox()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxCheckBox()`

See **external documentation**.

`new(Parent, Id, Label) -> wxCheckBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Label, [])*.

`new(Parent, Id, Label, Option::[Option]) -> wxCheckBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

`create(This, Parent, Id, Label) -> boolean()`

Types:

```
This = wxCheckBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Label, [])*.

create(This, Parent, Id, Label, Option::[Option]) -> boolean()

Types:

This = wxCheckBox()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Label = chardata() (see module unicode)

Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {validator, wx_object() (see module wx)}

See **external documentation**.

getValue(This) -> boolean()

Types:

This = wxCheckBox()

See **external documentation**.

get3StateValue(This) -> wx_enum() (see module wx)

Types:

This = wxCheckBox()

See **external documentation**.
Res = ?wxCHK_UNCHECKED | ?wxCHK_CHECKED | ?wxCHK_UNDETERMINED

is3rdStateAllowedForUser(This) -> boolean()

Types:

This = wxCheckBox()

See **external documentation**.

is3State(This) -> boolean()

Types:

This = wxCheckBox()

See **external documentation**.

isChecked(This) -> boolean()

Types:

This = wxCheckBox()

See **external documentation**.

setValue(This, State) -> ok

Types:

This = wxCheckBox()

State = boolean()

See **external documentation**.

set3StateValue(This, State) -> ok

Types:

This = wxCheckBox()

State = wx_enum() (see module wx)

See **external documentation**.
State = ?wxCHK_UNCHECKED | ?wxCHK_CHECKED | ?wxCHK_UNDETERMINED

destroy(This::wxCheckBox()) -> ok

Destroys this object, do not use object again

wxCheckListBox

Erlang module

See external documentation: **wxCheckListBox**.

This class is derived (and can use functions) from:
wxListBox
wxControlWithItems
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxCheckListBox()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxCheckListBox()`

See **external documentation**.

`new(Parent, Id) -> wxCheckListBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxCheckListBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {choices, [chardata() (see module unicode)]} | {style,
integer()} | {validator, wx_object() (see module wx)}
```

See **external documentation**.

`check(This, Index) -> ok`

Types:

```
This = wxCheckListBox()
Index = integer()
```

Equivalent to *check(This, Index, [])*.

check(This, Index, Option::[Option]) -> ok

Types:

```
This = wxCheckListBox()
Index = integer()
Option = {check, boolean()}
```

See **external documentation**.

isChecked(This, Index) -> boolean()

Types:

```
This = wxCheckListBox()
Index = integer()
```

See **external documentation**.

destroy(This::wxCheckListBox()) -> ok

Destroys this object, do not use object again

wxChildFocusEvent

Erlang module

See external documentation: **wxChildFocusEvent**.

Use *wxEvtHandler:connect/3* with EventType:

child_focus

See also the message variant *#wxChildFocus{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxChildFocusEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getWindow(This) -> wxWindow() (see module wxWindow)`

Types:

`This = wxChildFocusEvent()`

See **external documentation**.

wxChoice

Erlang module

See external documentation: **wxChoice**.

This class is derived (and can use functions) from:
wxControlWithItems
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxChoice()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxChoice()`

See **external documentation**.

`new(Parent, Id) -> wxChoice()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxChoice()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {choices, [chardata() (see module unicode)]} | {style,
integer()} | {validator, wx_object() (see module wx)}
```

See **external documentation**.

`create(This, Parent, Id, Pos, Size, Choices) -> boolean()`

Types:

```
This = wxChoice()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
```

Equivalent to *create(This, Parent, Id, Pos, Size, Choices, [])*.

create(This, Parent, Id, Pos, Size, Choices, Option::[Option]) -> boolean()

Types:

```
This = wxChoice()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
Option = {style, integer()} | {validator, wx_object() (see module wx)}
```

See **external documentation**.

delete(This, N) -> ok

Types:

```
This = wxChoice()
N = integer()
```

See **external documentation**.

getColumns(This) -> integer()

Types:

```
This = wxChoice()
```

See **external documentation**.

setColumns(This) -> ok

Types:

```
This = wxChoice()
```

Equivalent to *setColumns(This, [])*.

setColumns(This, Option::[Option]) -> ok

Types:

```
This = wxChoice()
Option = {n, integer()}
```

See **external documentation**.

destroy(This::wxChoice()) -> ok

Destroys this object, do not use object again

wxChoicebook

Erlang module

See external documentation: **wxChoicebook**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxChoicebook()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxChoicebook()`

See **external documentation**.

`new(Parent, Id) -> wxChoicebook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxChoicebook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`addPage(This, Page, Text) -> boolean()`

Types:

```
This = wxChoicebook()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
```

Equivalent to *addPage(This, Page, Text, [])*.

`addPage(This, Page, Text, Option::[Option]) -> boolean()`

Types:

```
This = wxChoicebook()
```

```
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}
```

See **external documentation**.

advanceSelection(This) -> ok

Types:

```
This = wxChoicebook()
```

Equivalent to *advanceSelection(This, [])*.

advanceSelection(This, Option::[Option]) -> ok

Types:

```
This = wxChoicebook()
Option = {forward, boolean()}
```

See **external documentation**.

assignImageList(This, ImageList) -> ok

Types:

```
This = wxChoicebook()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

create(This, Parent, Id) -> boolean()

Types:

```
This = wxChoicebook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

create(This, Parent, Id, Option::[Option]) -> boolean()

Types:

```
This = wxChoicebook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

deleteAllPages(This) -> boolean()

Types:

```
This = wxChoicebook()
```

See **external documentation**.

deletePage(This, N) -> boolean()

Types:

This = wxChoicebook()

N = integer()

See **external documentation**.

removePage(This, N) -> boolean()

Types:

This = wxChoicebook()

N = integer()

See **external documentation**.

getCurrentPage(This) -> wxWindow() (see module wxWindow)

Types:

This = wxChoicebook()

See **external documentation**.

getImageList(This) -> wxImageList() (see module wxImageList)

Types:

This = wxChoicebook()

See **external documentation**.

getPage(This, N) -> wxWindow() (see module wxWindow)

Types:

This = wxChoicebook()

N = integer()

See **external documentation**.

getPageCount(This) -> integer()

Types:

This = wxChoicebook()

See **external documentation**.

getPageImage(This, N) -> integer()

Types:

This = wxChoicebook()

N = integer()

See **external documentation**.

getPageText(This, N) -> charlist() (see module unicode)

Types:

This = wxChoicebook()

N = integer()

See **external documentation**.

getSelection(This) -> integer()

Types:

This = wxChoicebook()

See **external documentation**.

hitTest(This, Pt) -> Result

Types:

Result = {Res::integer(), Flags::integer()}
This = wxChoicebook()
Pt = {X::integer(), Y::integer()}

See **external documentation**.

insertPage(This, N, Page, Text) -> boolean()

Types:

This = wxChoicebook()
N = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)

Equivalent to *insertPage(This, N, Page, Text, [])*.

insertPage(This, N, Page, Text, Option::[Option]) -> boolean()

Types:

This = wxChoicebook()
N = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}

See **external documentation**.

setImageList(This, ImageList) -> ok

Types:

This = wxChoicebook()
ImageList = wxImageList() (see module wxImageList)

See **external documentation**.

setPageSize(This, Size) -> ok

Types:

This = wxChoicebook()
Size = {W::integer(), H::integer()}

See **external documentation**.

setPageImage(This, N, ImageId) -> boolean()

Types:

This = wxChoicebook()
N = integer()
ImageId = integer()

See **external documentation**.

setPageText(This, N, StrText) -> boolean()

Types:

This = wxChoicebook()
N = integer()
StrText = chardata() (see module unicode)

See **external documentation**.

setSelection(This, N) -> integer()

Types:

This = wxChoicebook()
N = integer()

See **external documentation**.

changeSelection(This, N) -> integer()

Types:

This = wxChoicebook()
N = integer()

See **external documentation**.

destroy(This::wxChoicebook()) -> ok

Destroys this object, do not use object again

wxClientDC

Erlang module

See external documentation: **wxClientDC**.

This class is derived (and can use functions) from:
wxWindowDC
wxDC

DATA TYPES

`wxClientDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxClientDC()`

See **external documentation**.

`new(Win) -> wxClientDC()`

Types:

`Win = wxWindow() (see module wxWindow)`

See **external documentation**.

`destroy(This::wxClientDC()) -> ok`

Destroys this object, do not use object again

wxClipboard

Erlang module

See external documentation: **wxClipboard**.

DATA TYPES

`wxClipboard()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxClipboard()`

See **external documentation**.

`addData(This, Data) -> boolean()`

Types:

```
This = wxClipboard()
Data = wxDataObject() (see module wxDataObject)
```

See **external documentation**.

`clear(This) -> ok`

Types:

```
This = wxClipboard()
```

See **external documentation**.

`close(This) -> ok`

Types:

```
This = wxClipboard()
```

See **external documentation**.

`flush(This) -> boolean()`

Types:

```
This = wxClipboard()
```

See **external documentation**.

`getData(This, Data) -> boolean()`

Types:

```
This = wxClipboard()
Data = wxDataObject() (see module wxDataObject)
```

See **external documentation**.

isOpened(This) -> boolean()

Types:

This = wxClipboard()

See **external documentation**.

open(This) -> boolean()

Types:

This = wxClipboard()

See **external documentation**.

setData(This, Data) -> boolean()

Types:

This = wxClipboard()

Data = wxDataObject() (see module wxDataObject)

See **external documentation**.

usePrimarySelection(This) -> ok

Types:

This = wxClipboard()

Equivalent to *usePrimarySelection(This, [])*.

usePrimarySelection(This, Option::[Option]) -> ok

Types:

This = wxClipboard()

Option = {primary, boolean()}

See **external documentation**.

isSupported(This, Format) -> boolean()

Types:

This = wxClipboard()

Format = integer()

See **external documentation**.

get() -> wxClipboard()

See **external documentation**.

destroy(This::wxClipboard()) -> ok

Destroys this object, do not use object again

wxCloseEvent

Erlang module

See external documentation: **wxCloseEvent**.

Use *wxEvtHandler:connect/3* with EventType:

close_window, *end_session*, *query_end_session*

See also the message variant *#wxClose{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxCloseEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`canVeto(This) -> boolean()`

Types:

`This = wxCloseEvent()`

See **external documentation**.

`getLoggingOff(This) -> boolean()`

Types:

`This = wxCloseEvent()`

See **external documentation**.

`setCanVeto(This, CanVeto) -> ok`

Types:

`This = wxCloseEvent()`
`CanVeto = boolean()`

See **external documentation**.

`setLoggingOff(This, LogOff) -> ok`

Types:

`This = wxCloseEvent()`
`LogOff = boolean()`

See **external documentation**.

`veto(This) -> ok`

Types:

`This = wxCloseEvent()`

Equivalent to *veto(This, [])*.

veto(This, Option::[Option]) -> ok

Types:

This = wxCloseEvent()

Option = {veto, boolean()}

See **external documentation**.

wxColourData

Erlang module

See external documentation: **wxColourData**.

DATA TYPES

`wxColourData()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxColourData()`

See **external documentation**.

`new(Data) -> wxColourData()`

Types:

`Data = wxColourData()`

See **external documentation**.

`getChooseFull(This) -> boolean()`

Types:

`This = wxColourData()`

See **external documentation**.

`getColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxColourData()`

See **external documentation**.

`getCustomColour(This, I) -> wx_colour4() (see module wx)`

Types:

`This = wxColourData()`
`I = integer()`

See **external documentation**.

`setChooseFull(This, Flag) -> ok`

Types:

`This = wxColourData()`
`Flag = boolean()`

See **external documentation**.

setColour(This, Colour) -> ok

Types:

```
This = wxColourData()
Colour = wx_colour() (see module wx)
```

See **external documentation**.

setCustomColour(This, I, Colour) -> ok

Types:

```
This = wxColourData()
I = integer()
Colour = wx_colour() (see module wx)
```

See **external documentation**.

destroy(This::wxColourData()) -> ok

Destroys this object, do not use object again

wxColourDialog

Erlang module

See external documentation: **wxColourDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxColourDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxColourDialog()`

See **external documentation**.

`new(Parent) -> wxColourDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxColourDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {data, wxColourData() (see module wxColourData)}
```

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

```
This = wxColourDialog()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

```
This = wxColourDialog()
Parent = wxWindow() (see module wxWindow)
Option = {data, wxColourData() (see module wxColourData)}
```

See **external documentation**.

getColourData(This) -> wxColourData() (see module wxColourData)

Types:

This = wxColourDialog()

See **external documentation**.

destroy(This::wxColourDialog()) -> ok

Destroys this object, do not use object again

wxColourPickerCtrl

Erlang module

See external documentation: **wxColourPickerCtrl**.

This class is derived (and can use functions) from:
wxPickerBase
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxColourPickerCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxColourPickerCtrl()`

See **external documentation**.

`new(Parent, Id) -> wxColourPickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxColourPickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {col, wx_colour() (see module wx)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
| {validator, wx_object() (see module wx)}
```

See **external documentation**.

`create(This, Parent, Id) -> boolean()`

Types:

```
This = wxColourPickerCtrl()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

create(This, Parent, Id, Option::[Option]) -> boolean()

Types:

This = wxColourPickerCtrl()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Option = {col, wx_colour() (see module wx)} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {validator, wx_object() (see module wx)}

See **external documentation**.

getColour(This) -> wx_colour4() (see module wx)

Types:

This = wxColourPickerCtrl()

See **external documentation**.

setColour(This, Text) -> boolean()

Types:

This = wxColourPickerCtrl()

Text = chardata() (see module unicode)

See **external documentation**.
Also:
setColour(This, Col) -> ok when
This::wxColourPickerCtrl(), Col::wx:wx_colour().

destroy(This::wxColourPickerCtrl()) -> ok

Destroys this object, do not use object again

wxColourPickerEvent

Erlang module

See external documentation: **wxColourPickerEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_colourpicker_changed

See also the message variant *#wxColourPicker{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxColourPickerEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxColourPickerEvent()`

See **external documentation**.

wxComboBox

Erlang module

See external documentation: **wxComboBox**.

This class is derived (and can use functions) from:
wxControlWithItems
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxComboBox()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxComboBox()`

See **external documentation**.

`new(Parent, Id) -> wxComboBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxComboBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {value, chardata() (see module unicode)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {choices,
[chardata() (see module unicode)]} | {style, integer()} | {validator,
wx_object() (see module wx)}
```

See **external documentation**.

`create(This, Parent, Id, Value, Pos, Size, Choices) -> boolean()`

Types:

```
This = wxComboBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Value = chardata() (see module unicode)
Pos = {X::integer(), Y::integer()}
```

```
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
```

Equivalent to *create(This, Parent, Id, Value, Pos, Size, Choices, [])*.

create(This, Parent, Id, Value, Pos, Size, Choices, Option::[Option]) -> boolean()

Types:

```
This = wxComboBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Value = chardata() (see module unicode)
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
Option = {style, integer()} | {validator, wx_object() (see module wx)}
```

See **external documentation**.

canCopy(This) -> boolean()

Types:

```
This = wxComboBox()
```

See **external documentation**.

canCut(This) -> boolean()

Types:

```
This = wxComboBox()
```

See **external documentation**.

canPaste(This) -> boolean()

Types:

```
This = wxComboBox()
```

See **external documentation**.

canRedo(This) -> boolean()

Types:

```
This = wxComboBox()
```

See **external documentation**.

canUndo(This) -> boolean()

Types:

```
This = wxComboBox()
```

See **external documentation**.

copy(This) -> ok

Types:

This = wxComboBox()

See **external documentation**.

cut(This) -> ok

Types:

This = wxComboBox()

See **external documentation**.

getInsertionPoint(This) -> integer()

Types:

This = wxComboBox()

See **external documentation**.

getLastPosition(This) -> integer()

Types:

This = wxComboBox()

See **external documentation**.

getValue(This) -> charlist() (see module unicode)

Types:

This = wxComboBox()

See **external documentation**.

paste(This) -> ok

Types:

This = wxComboBox()

See **external documentation**.

redo(This) -> ok

Types:

This = wxComboBox()

See **external documentation**.

replace(This, From, To, Value) -> ok

Types:

This = wxComboBox()
From = integer()
To = integer()
Value = chardata() (see module unicode)

See **external documentation**.

remove(This, From, To) -> ok

Types:

This = wxComboBox()
From = integer()
To = integer()

See **external documentation**.

setInsertionPoint(This, Pos) -> ok

Types:

This = wxComboBox()
Pos = integer()

See **external documentation**.

setInsertionPointEnd(This) -> ok

Types:

This = wxComboBox()

See **external documentation**.

setSelection(This, N) -> ok

Types:

This = wxComboBox()
N = integer()

See **external documentation**.

setSelection(This, From, To) -> ok

Types:

This = wxComboBox()
From = integer()
To = integer()

See **external documentation**.

setValue(This, Value) -> ok

Types:

This = wxComboBox()
Value = chardata() (see module unicode)

See **external documentation**.

undo(This) -> ok

Types:

This = wxComboBox()

See **external documentation**.

destroy(This::wxComboBox()) -> ok

Destroys this object, do not use object again

wxCommandEvent

Erlang module

See external documentation: **wxCommandEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_button_clicked, *command_checkbox_clicked*, *command_choice_selected*, *command_listbox_selected*, *command_listbox_doubleclicked*, *command_text_updated*, *command_text_enter*, *command_menu_selected*, *command_slider_updated*, *command_radiobox_selected*, *command_radiobutton_selected*, *command_scrollbar_updated*, *command_vlbox_selected*, *command_combobox_selected*, *command_tool_rclicked*, *command_tool_enter*, *command_checklistbox_toggled*, *command_togglebutton_clicked*, *command_left_click*, *command_left_dclick*, *command_right_click*, *command_set_focus*, *command_kill_focus*, *command_enter*

See also the message variant *#wxCommand{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxCommandEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getClientData(This) -> term()`

Types:

`This = wxCommandEvent()`

See **external documentation**.

`getExtraLong(This) -> integer()`

Types:

`This = wxCommandEvent()`

See **external documentation**.

`getInt(This) -> integer()`

Types:

`This = wxCommandEvent()`

See **external documentation**.

`getSelection(This) -> integer()`

Types:

`This = wxCommandEvent()`

See **external documentation**.

getString(This) -> charlist() (see module unicode)

Types:

This = wxCommandEvent()

See **external documentation**.

isChecked(This) -> boolean()

Types:

This = wxCommandEvent()

See **external documentation**.

isSelection(This) -> boolean()

Types:

This = wxCommandEvent()

See **external documentation**.

setInt(This, I) -> ok

Types:

This = wxCommandEvent()

I = integer()

See **external documentation**.

setString(This, S) -> ok

Types:

This = wxCommandEvent()

S = chardata() (see module unicode)

See **external documentation**.

wxContextMenuEvent

Erlang module

See external documentation: **wxContextMenuEvent**.

Use *wxEvtHandler:connect/3* with EventType:

context_menu

See also the message variant *#wxContextMenu{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxContextMenuEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getPosition(This) -> {X::integer(), Y::integer()}`

Types:

`This = wxContextMenuEvent()`

See **external documentation**.

`setPosition(This, Pos) -> ok`

Types:

`This = wxContextMenuEvent()`
`Pos = {X::integer(), Y::integer()}`

See **external documentation**.

wxControl

Erlang module

See external documentation: **wxControl**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxControl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getLabel(This) -> charlist() (see module unicode)`

Types:

`This = wxControl()`

See **external documentation**.

`setLabel(This, Label) -> ok`

Types:

`This = wxControl()`
`Label = chardata() (see module unicode)`

See **external documentation**.

wxControlWithItems

Erlang module

See external documentation: **wxControlWithItems**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxControlWithItems()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`append(This, Item) -> integer()`

Types:

```
This = wxControlWithItems()
Item = chardata() (see module unicode)
```

See **external documentation**.

`append(This, Item, ClientData) -> integer()`

Types:

```
This = wxControlWithItems()
Item = chardata() (see module unicode)
ClientData = term()
```

See **external documentation**.

`appendStrings(This, Strings) -> ok`

Types:

```
This = wxControlWithItems()
Strings = [chardata() (see module unicode)]
```

See **external documentation**.

`clear(This) -> ok`

Types:

```
This = wxControlWithItems()
```

See **external documentation**.

`delete(This, N) -> ok`

Types:

```
This = wxControlWithItems()
N = integer()
```

See **external documentation**.

```
findString(This, S) -> integer()
```

Types:

```
This = wxControlWithItems()
S = chardata() (see module unicode)
```

Equivalent to *findString(This, S, [])*.

```
findString(This, S, Option::[Option]) -> integer()
```

Types:

```
This = wxControlWithItems()
S = chardata() (see module unicode)
Option = {bCase, boolean()}
```

See **external documentation**.

```
getClientData(This, N) -> term()
```

Types:

```
This = wxControlWithItems()
N = integer()
```

See **external documentation**.

```
setClientData(This, N, ClientData) -> ok
```

Types:

```
This = wxControlWithItems()
N = integer()
ClientData = term()
```

See **external documentation**.

```
getCount(This) -> integer()
```

Types:

```
This = wxControlWithItems()
```

See **external documentation**.

```
getSelection(This) -> integer()
```

Types:

```
This = wxControlWithItems()
```

See **external documentation**.

```
getString(This, N) -> charlist() (see module unicode)
```

Types:

```
This = wxControlWithItems()
```

```
N = integer()
```

See **external documentation**.

getStringSelection(This) -> charlist() (see module unicode)

Types:

```
This = wxControlWithItems()
```

See **external documentation**.

insert(This, Item, Pos) -> integer()

Types:

```
This = wxControlWithItems()
Item = chardata() (see module unicode)
Pos = integer()
```

See **external documentation**.

insert(This, Item, Pos, ClientData) -> integer()

Types:

```
This = wxControlWithItems()
Item = chardata() (see module unicode)
Pos = integer()
ClientData = term()
```

See **external documentation**.

isEmpty(This) -> boolean()

Types:

```
This = wxControlWithItems()
```

See **external documentation**.

select(This, N) -> ok

Types:

```
This = wxControlWithItems()
N = integer()
```

See **external documentation**.

setSelection(This, N) -> ok

Types:

```
This = wxControlWithItems()
N = integer()
```

See **external documentation**.

setString(This, N, S) -> ok

Types:

```
This = wxControlWithItems()
```

```
N = integer()
S = chardata() (see module unicode)
```

See **external documentation**.

setStringSelection(This, S) -> boolean()

Types:

```
This = wxControlWithItems()
S = chardata() (see module unicode)
```

See **external documentation**.

wxCursor

Erlang module

See external documentation: **wxCursor**.

This class is derived (and can use functions) from:
wxBitmap

DATA TYPES

`wxCursor()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxCursor()`

See **external documentation**.

`new(CursorId) -> wxCursor()`

Types:

`CursorId = integer()`

See **external documentation**.
Also:
new(Image) -> wxCursor() when
Image::wxImage:wxImage().

`new(Bits, Width, Height) -> wxCursor()`

Types:

`Bits = binary()`
`Width = integer()`
`Height = integer()`

Equivalent to *new(Bits, Width, Height, [])*.

`new(Bits, Width, Height, Option::[Option]) -> wxCursor()`

Types:

`Bits = binary()`
`Width = integer()`
`Height = integer()`
`Option = {hotSpotX, integer()} | {hotSpotY, integer()}`

See **external documentation**.

`ok(This) -> boolean()`

Types:

`This = wxCursor()`

See **external documentation**.

`destroy(This::wxCursor()) -> ok`

Destroys this object, do not use object again

wxDC

Erlang module

See external documentation: **wxDC**.

DATA TYPES

`wxDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`blit(This, DestPt, Sz, Source, SrcPt) -> boolean()`

Types:

```
This = wxDC()
DestPt = {X::integer(), Y::integer()}
Sz = {W::integer(), H::integer()}
Source = wxDC()
SrcPt = {X::integer(), Y::integer()}
```

Equivalent to *blit(This, DestPt, Sz, Source, SrcPt, [])*.

`blit(This, DestPt, Sz, Source, SrcPt, Option::[Option]) -> boolean()`

Types:

```
This = wxDC()
DestPt = {X::integer(), Y::integer()}
Sz = {W::integer(), H::integer()}
Source = wxDC()
SrcPt = {X::integer(), Y::integer()}
Option = {rop, integer()} | {useMask, boolean()} | {srcPtMask,
{X::integer(), Y::integer()}}
```

See **external documentation**.

`calcBoundingBox(This, X, Y) -> ok`

Types:

```
This = wxDC()
X = integer()
Y = integer()
```

See **external documentation**.

`clear(This) -> ok`

Types:

```
This = wxDC()
```

See **external documentation**.

computeScaleAndOrigin(This) -> ok

Types:

This = wxDC()

See **external documentation**.

crossHair(This, Pt) -> ok

Types:

This = wxDC()

Pt = {X::integer(), Y::integer()}

See **external documentation**.

destroyClippingRegion(This) -> ok

Types:

This = wxDC()

See **external documentation**.

deviceToLogicalX(This, X) -> integer()

Types:

This = wxDC()

X = integer()

See **external documentation**.

deviceToLogicalXRel(This, X) -> integer()

Types:

This = wxDC()

X = integer()

See **external documentation**.

deviceToLogicalY(This, Y) -> integer()

Types:

This = wxDC()

Y = integer()

See **external documentation**.

deviceToLogicalYRel(This, Y) -> integer()

Types:

This = wxDC()

Y = integer()

See **external documentation**.

drawArc(This, Pt1, Pt2, Centre) -> ok

Types:

```
This = wxDC()
Pt1 = {X::integer(), Y::integer()}
Pt2 = {X::integer(), Y::integer()}
Centre = {X::integer(), Y::integer()}
```

See **external documentation**.

drawBitmap(This, Bmp, Pt) -> ok

Types:

```
This = wxDC()
Bmp = wxBitmap() (see module wxBitmap)
Pt = {X::integer(), Y::integer()}
```

Equivalent to *drawBitmap(This, Bmp, Pt, [])*.

drawBitmap(This, Bmp, Pt, Option::[Option]) -> ok

Types:

```
This = wxDC()
Bmp = wxBitmap() (see module wxBitmap)
Pt = {X::integer(), Y::integer()}
Option = {useMask, boolean()}
```

See **external documentation**.

drawCheckMark(This, Rect) -> ok

Types:

```
This = wxDC()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

See **external documentation**.

drawCircle(This, Pt, Radius) -> ok

Types:

```
This = wxDC()
Pt = {X::integer(), Y::integer()}
Radius = integer()
```

See **external documentation**.

drawEllipse(This, Rect) -> ok

Types:

```
This = wxDC()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

See **external documentation**.

```
drawEllipse(This, Pt, Sz) -> ok
```

Types:

```
This = wxDC()
Pt = {X::integer(), Y::integer()}
Sz = {W::integer(), H::integer()}
```

See **external documentation**.

```
drawEllipticArc(This, Pt, Sz, Sa, Ea) -> ok
```

Types:

```
This = wxDC()
Pt = {X::integer(), Y::integer()}
Sz = {W::integer(), H::integer()}
Sa = number()
Ea = number()
```

See **external documentation**.

```
drawIcon(This, Icon, Pt) -> ok
```

Types:

```
This = wxDC()
Icon = wxIcon() (see module wxIcon)
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

```
drawLabel(This, Text, Rect) -> ok
```

Types:

```
This = wxDC()
Text = chardata() (see module unicode)
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

Equivalent to *drawLabel(This, Text, Rect, [])*.

```
drawLabel(This, Text, Rect, Option::[Option]) -> ok
```

Types:

```
This = wxDC()
Text = chardata() (see module unicode)
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
Option = {alignment, integer()} | {indexAccel, integer()}
```

See **external documentation**.

```
drawLine(This, Pt1, Pt2) -> ok
```

Types:

```
This = wxDC()
Pt1 = {X::integer(), Y::integer()}
Pt2 = {X::integer(), Y::integer()}
```

See **external documentation**.

drawLines(This, Points) -> ok

Types:

```
This = wxDC()
Points = [{X::integer(), Y::integer()}]
```

Equivalent to *drawLines(This, Points, [])*.

drawLines(This, Points, Option::[Option]) -> ok

Types:

```
This = wxDC()
Points = [{X::integer(), Y::integer()}]
Option = {xoffset, integer()} | {yoffset, integer()}
```

See **external documentation**.

drawPolygon(This, Points) -> ok

Types:

```
This = wxDC()
Points = [{X::integer(), Y::integer()}]
```

Equivalent to *drawPolygon(This, Points, [])*.

drawPolygon(This, Points, Option::[Option]) -> ok

Types:

```
This = wxDC()
Points = [{X::integer(), Y::integer()}]
Option = {xoffset, integer()} | {yoffset, integer()} | {fillStyle,
integer()}
```

See **external documentation**.

drawPoint(This, Pt) -> ok

Types:

```
This = wxDC()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

drawRectangle(This, Rect) -> ok

Types:

```
This = wxDC()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

See **external documentation**.

drawRectangle(This, Pt, Sz) -> ok

Types:

```
This = wxDC()
Pt = {X::integer(), Y::integer()}
Sz = {W::integer(), H::integer()}
```

See **external documentation**.

drawRotatedText(This, Text, Pt, Angle) -> ok

Types:

```
This = wxDC()
Text = chardata() (see module unicode)
Pt = {X::integer(), Y::integer()}
Angle = number()
```

See **external documentation**.

drawRoundedRectangle(This, R, Radius) -> ok

Types:

```
This = wxDC()
R = {X::integer(), Y::integer(), W::integer(), H::integer()}
Radius = number()
```

See **external documentation**.

drawRoundedRectangle(This, Pt, Sz, Radius) -> ok

Types:

```
This = wxDC()
Pt = {X::integer(), Y::integer()}
Sz = {W::integer(), H::integer()}
Radius = number()
```

See **external documentation**.

drawText(This, Text, Pt) -> ok

Types:

```
This = wxDC()
Text = chardata() (see module unicode)
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

endDoc(This) -> ok

Types:

```
This = wxDC()
```

See **external documentation**.

endPage(This) -> ok

Types:

```
This = wxDC()
```

See **external documentation**.

floodFill(This, Pt, Col) -> boolean()

Types:

```
This = wxDC()
Pt = {X::integer(), Y::integer()}
Col = wx_colour() (see module wx)
```

Equivalent to *floodFill(This, Pt, Col, [])*.

floodFill(This, Pt, Col, Option::[Option]) -> boolean()

Types:

```
This = wxDC()
Pt = {X::integer(), Y::integer()}
Col = wx_colour() (see module wx)
Option = {style, integer()}
```

See **external documentation**.

getBackground(This) -> wxBrush() (see module wxBrush)

Types:

```
This = wxDC()
```

See **external documentation**.

getBackgroundMode(This) -> integer()

Types:

```
This = wxDC()
```

See **external documentation**.

getBrush(This) -> wxBrush() (see module wxBrush)

Types:

```
This = wxDC()
```

See **external documentation**.

getCharHeight(This) -> integer()

Types:

```
This = wxDC()
```

See **external documentation**.

getCharWidth(This) -> integer()

Types:

```
This = wxDC()
```

See **external documentation**.

getClippingBox(This) -> Result

Types:

Result = {X::integer(), Y::integer(), W::integer(), H::integer()}

This = wxDC()

See **external documentation**.

getFont(This) -> wxFont() (see module wxFont)

Types:

This = wxDC()

See **external documentation**.

getLayoutDirection(This) -> wx_enum() (see module wx)

Types:

This = wxDC()

See **external documentation**.
Res = ?wxLayout_Default | ?wxLayout_LeftToRight | ?wxLayout_RightToLeft

getLogicalFunction(This) -> integer()

Types:

This = wxDC()

See **external documentation**.

getMapMode(This) -> integer()

Types:

This = wxDC()

See **external documentation**.

getMultiLineTextExtent(This, String) -> {W::integer(), H::integer()}

Types:

This = wxDC()

String = chardata() (see module unicode)

See **external documentation**.

getMultiLineTextExtent(This, String, Option::[Option]) -> {Width::integer(), Height::integer(), HeightLine::integer()}

Types:

This = wxDC()

String = chardata() (see module unicode)

Option = {font, wxFont() (see module wxFont)}

See **external documentation**.

getPartialTextExtents(This, Text) -> Result

Types:

```
Result = {Res::boolean(), Widths::[integer()]}
This = wxDC()
Text = chardata() (see module unicode)
```

See **external documentation**.

```
getPen(This) -> wxPen() (see module wxPen)
```

Types:

```
This = wxDC()
```

See **external documentation**.

```
getPixel(This, Pt) -> Result
```

Types:

```
Result = {Res::boolean(), Col::wx_colour4() (see module wx)}
This = wxDC()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

```
getPPI(This) -> {W::integer(), H::integer()}
```

Types:

```
This = wxDC()
```

See **external documentation**.

```
getSize(This) -> {W::integer(), H::integer()}
```

Types:

```
This = wxDC()
```

See **external documentation**.

```
getSizeMM(This) -> {W::integer(), H::integer()}
```

Types:

```
This = wxDC()
```

See **external documentation**.

```
getTextBackground(This) -> wx_colour4() (see module wx)
```

Types:

```
This = wxDC()
```

See **external documentation**.

```
getTextExtent(This, String) -> {W::integer(), H::integer()}
```

Types:

```
This = wxDC()
String = chardata() (see module unicode)
```

See **external documentation**.

getTextExtent(This, String, Option::[Option]) -> Result

Types:

Result = {X::integer(), Y::integer(), Descent::integer(), ExternalLeading::integer()}

This = wxDC()

String = chardata() (see module unicode)

Option = {theFont, wxFont() (see module wxFont)}

See **external documentation**.

getTextForeground(This) -> wx_colour4() (see module wx)

Types:

This = wxDC()

See **external documentation**.

getUserScale(This) -> {X::number(), Y::number()}

Types:

This = wxDC()

See **external documentation**.

gradientFillConcentric(This, Rect, InitialColour, DestColour) -> ok

Types:

This = wxDC()

Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}

InitialColour = wx_colour() (see module wx)

DestColour = wx_colour() (see module wx)

See **external documentation**.

gradientFillConcentric(This, Rect, InitialColour, DestColour, CircleCenter) -> ok

Types:

This = wxDC()

Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}

InitialColour = wx_colour() (see module wx)

DestColour = wx_colour() (see module wx)

CircleCenter = {X::integer(), Y::integer()}

See **external documentation**.

gradientFillLinear(This, Rect, InitialColour, DestColour) -> ok

Types:

This = wxDC()

Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}

InitialColour = wx_colour() (see module wx)

DestColour = wx_colour() (see module wx)

Equivalent to *gradientFillLinear(This, Rect, InitialColour, DestColour, [])*.

gradientFillLinear(This, Rect, InitialColour, DestColour, Option::[Option]) -> ok

Types:

This = wxDC()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
InitialColour = wx_colour() (see module wx)
DestColour = wx_colour() (see module wx)
Option = {nDirection, wx_enum() (see module wx)}

See **external documentation**.
NDirection = ?wxLEFT | ?wxRIGHT | ?wxUP | ?wxDOWN | ?wxTOP | ?wxBOTTOM | ?wxNORTH | ?wxSOUTH | ?wxWEST | ?wxEAST | ?wxALL

logicalToDeviceX(This, X) -> integer()

Types:

This = wxDC()
X = integer()

See **external documentation**.

logicalToDeviceXRel(This, X) -> integer()

Types:

This = wxDC()
X = integer()

See **external documentation**.

logicalToDeviceY(This, Y) -> integer()

Types:

This = wxDC()
Y = integer()

See **external documentation**.

logicalToDeviceYRel(This, Y) -> integer()

Types:

This = wxDC()
Y = integer()

See **external documentation**.

maxX(This) -> integer()

Types:

This = wxDC()

See **external documentation**.

maxY(This) -> integer()

Types:

This = wxDC()

See **external documentation**.

minX(This) -> integer()

Types:

This = wxDC()

See **external documentation**.

minY(This) -> integer()

Types:

This = wxDC()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxDC()

See **external documentation**.

resetBoundingBox(This) -> ok

Types:

This = wxDC()

See **external documentation**.

setAxisOrientation(This, XLeftRight, YBottomUp) -> ok

Types:

This = wxDC()
XLeftRight = boolean()
YBottomUp = boolean()

See **external documentation**.

setBackground(This, Brush) -> ok

Types:

This = wxDC()
Brush = wxBrush() (see module wxBrush)

See **external documentation**.

setBackgroundMode(This, Mode) -> ok

Types:

This = wxDC()
Mode = integer()

See **external documentation**.

setBrush(This, Brush) -> ok

Types:

This = wxDC()
Brush = wxBrush() (see module wxBrush)

See **external documentation**.

setClippingRegion(This, Region) -> ok

Types:

This = wxDC()
Region = wxRegion() (see module wxRegion)

See **external documentation**.
Also:
setClippingRegion(This, Rect) -> ok when
This::wxDC(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

setClippingRegion(This, Pt, Sz) -> ok

Types:

This = wxDC()
Pt = {X::integer(), Y::integer()}
Sz = {W::integer(), H::integer()}

See **external documentation**.

setDeviceOrigin(This, X, Y) -> ok

Types:

This = wxDC()
X = integer()
Y = integer()

See **external documentation**.

setFont(This, Font) -> ok

Types:

This = wxDC()
Font = wxFont() (see module wxFont)

See **external documentation**.

setLayoutDirection(This, Dir) -> ok

Types:

This = wxDC()
Dir = wx_enum() (see module wx)

See **external documentation**.
Dir = ?wxLayout_Default | ?wxLayout_LeftToRight | ?wxLayout_RightToLeft

setLogicalFunction(This, Function) -> ok

Types:

```
This = wxDC()
Function = integer()
```

See **external documentation**.

setMapMode(This, Mode) -> ok

Types:

```
This = wxDC()
Mode = integer()
```

See **external documentation**.

setPalette(This, Palette) -> ok

Types:

```
This = wxDC()
Palette = wxPalette() (see module wxPalette)
```

See **external documentation**.

setPen(This, Pen) -> ok

Types:

```
This = wxDC()
Pen = wxPen() (see module wxPen)
```

See **external documentation**.

setTextBackground(This, Colour) -> ok

Types:

```
This = wxDC()
Colour = wx_colour() (see module wx)
```

See **external documentation**.

setTextForeground(This, Colour) -> ok

Types:

```
This = wxDC()
Colour = wx_colour() (see module wx)
```

See **external documentation**.

setUserScale(This, X, Y) -> ok

Types:

```
This = wxDC()
X = number()
Y = number()
```

See **external documentation**.

startDoc(This, Message) -> boolean()

Types:

```
This = wxDC()
Message = chardata() (see module unicode)
```

See **external documentation**.

`startPage(This) -> ok`

Types:

```
This = wxDC()
```

See **external documentation**.

wxDataObject

Erlang module

See external documentation: **wxDataObject**.

DATA TYPES

`wxDataObject()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxDateEvent

Erlang module

See external documentation: **wxDateEvent**.

Use *wxEvtHandler:connect/3* with EventType:

date_changed

See also the message variant *#wxDate{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxDateEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getDate(This) -> wx_datetime() (see module wx)`

Types:

`This = wxDateEvent()`

See **external documentation**.

wxDatePickerCtrl

Erlang module

See external documentation: **wxDatePickerCtrl**.

This class is derived (and can use functions) from:
wxPickerBase
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxDatePickerCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxDatePickerCtrl()`

See **external documentation**.

`new(Parent, Id) -> wxDatePickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxDatePickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {date, wx_datetime() (see module wx)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
| {validator, wx_object() (see module wx)}
```

See **external documentation**.

`getRange(This, Dt1, Dt2) -> boolean()`

Types:

```
This = wxDatePickerCtrl()
Dt1 = wx_datetime() (see module wx)
Dt2 = wx_datetime() (see module wx)
```

See **external documentation**.

getValue(This) -> wx_datetime() (see module wx)

Types:

```
This = wxDatePickerCtrl()
```

See **external documentation**.

setRange(This, Dt1, Dt2) -> ok

Types:

```
This = wxDatePickerCtrl()
Dt1 = wx_datetime() (see module wx)
Dt2 = wx_datetime() (see module wx)
```

See **external documentation**.

setValue(This, Date) -> ok

Types:

```
This = wxDatePickerCtrl()
Date = wx_datetime() (see module wx)
```

See **external documentation**.

destroy(This::wxDatePickerCtrl()) -> ok

Destroys this object, do not use object again

wxDialog

Erlang module

See external documentation: **wxDialog**.

This class is derived (and can use functions) from:
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxDialog()`

See **external documentation**.

`new(Parent, Id, Title) -> wxDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Title, [])*.

`new(Parent, Id, Title, Option::[Option]) -> wxDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent, Id, Title) -> boolean()`

Types:

```
This = wxDialog()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Title, [])*.

create(This, Parent, Id, Title, Option::[Option]) -> boolean()

Types:

This = wxDialog()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Title = chardata() (see module unicode)

Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}

See **external documentation**.

createButtonSizer(This, Flags) -> wxSizer() (see module wxSizer)

Types:

This = wxDialog()

Flags = integer()

See **external documentation**.

createStdDialogButtonSizer(This, Flags) -> wxStdDialogButtonSizer() (see module wxStdDialogButtonSizer)

Types:

This = wxDialog()

Flags = integer()

See **external documentation**.

endModal(This, RetCode) -> ok

Types:

This = wxDialog()

RetCode = integer()

See **external documentation**.

getAffirmativeId(This) -> integer()

Types:

This = wxDialog()

See **external documentation**.

getReturnCode(This) -> integer()

Types:

This = wxDialog()

See **external documentation**.

isModal(This) -> boolean()

Types:

This = wxDialog()

See **external documentation**.

setAffirmativeId(This, AffirmativeId) -> ok

Types:

```
This = wxDialog()
AffirmativeId = integer()
```

See **external documentation**.

setReturnCode(This, ReturnCode) -> ok

Types:

```
This = wxDialog()
ReturnCode = integer()
```

See **external documentation**.

show(This) -> boolean()

Types:

```
This = wxDialog()
```

Equivalent to *show(This, [])*.

show(This, Option::[Option]) -> boolean()

Types:

```
This = wxDialog()
Option = {show, boolean()}
```

See **external documentation**.

showModal(This) -> integer()

Types:

```
This = wxDialog()
```

See **external documentation**.

destroy(This::wxDialog()) -> ok

Destroys this object, do not use object again

wxDirDialog

Erlang module

See external documentation: **wxDirDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxDirDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent) -> wxDirDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxDirDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`

`Option = {title, chardata() (see module unicode)} | {defaultPath, chardata() (see module unicode)} | {style, integer()} | {pos, {X::integer(), Y::integer()}} | {sz, {W::integer(), H::integer()}}`

See **external documentation**.

`getPath(This) -> charlist() (see module unicode)`

Types:

`This = wxDirDialog()`

See **external documentation**.

`getMessage(This) -> charlist() (see module unicode)`

Types:

`This = wxDirDialog()`

See **external documentation**.

`setMessage(This, Message) -> ok`

Types:

`This = wxDirDialog()`

```
Message = chardata() (see module unicode)
```

See **external documentation**.

setPath(This, Path) -> ok

Types:

```
This = wxDirDialog()
Path = chardata() (see module unicode)
```

See **external documentation**.

destroy(This::wxDirDialog()) -> ok

Destroys this object, do not use object again

wxDirPickerCtrl

Erlang module

See external documentation: **wxDirPickerCtrl**.

This class is derived (and can use functions) from:
wxPickerBase
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxDirPickerCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxDirPickerCtrl()`

See **external documentation**.

`new(Parent, Id) -> wxDirPickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxDirPickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {path, chardata() (see module unicode)} | {message, chardata()
(see module unicode)} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()} | {validator,
wx_object() (see module wx)}
```

See **external documentation**.

`create(This, Parent, Id) -> boolean()`

Types:

```
This = wxDirPickerCtrl()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

```
create(This, Parent, Id, Option::[Option]) -> boolean()
```

Types:

```
This = wxDirPickerCtrl()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {path, chardata() (see module unicode)} | {message, chardata()
(see module unicode)} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()} | {validator,
wx_object() (see module wx)}
```

See **external documentation**.

```
getPath(This) -> charlist() (see module unicode)
```

Types:

```
This = wxDirPickerCtrl()
```

See **external documentation**.

```
setPath(This, Str) -> ok
```

Types:

```
This = wxDirPickerCtrl()
Str = chardata() (see module unicode)
```

See **external documentation**.

```
destroy(This::wxDirPickerCtrl()) -> ok
```

Destroys this object, do not use object again

wxDisplayChangedEvent

Erlang module

See external documentation: **wxDisplayChangedEvent**.

Use *wxEvtHandler:connect/3* with EventType:

display_changed

See also the message variant *#wxDisplayChanged{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxDisplayChangedEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxEraseEvent

Erlang module

See external documentation: **wxEraseEvent**.

Use *wxEvtHandler:connect/3* with EventType:

erase_background

See also the message variant *#wxErase{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxEraseEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getDC(This) -> wxDC() (see module wxDC)`

Types:

`This = wxEraseEvent()`

See **external documentation**.

wxEvent

Erlang module

See external documentation: **wxEvent**.

DATA TYPES

`wxEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getId(This) -> integer()`

Types:

`This = wxEvent()`

See **external documentation**.

`getSkipped(This) -> boolean()`

Types:

`This = wxEvent()`

See **external documentation**.

`getTimestamp(This) -> integer()`

Types:

`This = wxEvent()`

See **external documentation**.

`isCommandEvent(This) -> boolean()`

Types:

`This = wxEvent()`

See **external documentation**.

`resumePropagation(This, PropagationLevel) -> ok`

Types:

`This = wxEvent()`
`PropagationLevel = integer()`

See **external documentation**.

`shouldPropagate(This) -> boolean()`

Types:

`This = wxEvent()`

See **external documentation**.

skip(This) -> ok

Types:

This = wxEvent()

Equivalent to *skip(This, [])*.

skip(This, Option::[Option]) -> ok

Types:

This = wxEvent()

Option = {skip, boolean()}

See **external documentation**.

stopPropagation(This) -> integer()

Types:

This = wxEvent()

See **external documentation**.

wxEvtHandler

Erlang module

The Event handler.

To get events from wxwidgets objects you subscribe to them by calling connect/[2-3]. Events are sent as messages, if no callback was supplied These messages will be *#wx{}* where EventRecord is a record that depends on the *event type*. The records are defined in: wx/include/wx.hrl.

If a callback was supplied to connect, the callback will be invoked (in another process) to handle the event. The callback should be of arity 2. fun(EventRecord::wx(), EventObject::wxObject()).

Beware that the callback will be in executed in new process each time.

The orginal documentation.

DATA TYPES

```
event() = wxAuiManager() | wxAuiNotebook() | wxCalendar() | wxChildFocus()
| wxClose() | wxColourPicker() | wxCommand() | wxContextMenu() | wxDate()
| wxDisplayChanged() | wxErase() | wxFileDirPicker() | wxFocus() |
wxFontPicker() | wxGrid() | wxHelp() | wxHtmlLink() | wxIconize() | wxIdle()
| wxJoystick() | wxKey() | wxList() | wxMaximize() | wxMenu() | wxMouse()
| wxMouseCaptureChanged() | wxMove() | wxNavigationKey() | wxNcPaint()
| wxNotebook() | wxPaint() | wxPaletteChanged() | wxQueryNewPalette()
| wxSash() | wxScroll() | wxScrollWin() | wxSetCursor() | wxShow() |
wxSize() | wxSpin() | wxSplitter() | wxStyledText() | wxSysColourChanged()
| wxTaskBarIcon() | wxTree() | wxUpdateUI() | wxWindowCreate() |
wxWindowDestroy()
wx() = #wx{id=undefined | integer(), obj=undefined | wx_object() (see module
wx), userData=undefined | term(), event=undefined | event()}
wxAuiManager() = #wxAuiManager{type=undefined | wxAuiManagerEventType(),
manager=undefined | wxAuiManager() (see module wxAuiManager), pane=undefined
| wxAuiPaneInfo() (see module wxAuiPaneInfo), button=undefined | integer(),
veto_flag=undefined | boolean(), canveto_flag=undefined | boolean(),
dc=undefined | wxDC() (see module wxDC)}
wxAuiManagerEventType() = aui_pane_button | aui_pane_close |
aui_pane_maximize | aui_pane_restore | aui_render | aui_find_manager
wxAuiNotebook() = #wxAuiNotebook{type=undefined | wxAuiNotebookEventType(),
old_selection=undefined | integer(), selection=undefined | integer(),
drag_source=undefined | wxAuiNotebook() (see module wxAuiNotebook)}
wxAuiNotebookEventType() = command_auinotebook_page_close |
command_auinotebook_page_changed | command_auinotebook_page_changing
| command_auinotebook_button | command_auinotebook_begin_drag |
command_auinotebook_end_drag | command_auinotebook_drag_motion |
command_auinotebook_allow_dnd | command_auinotebook_tab_middle_down |
command_auinotebook_tab_middle_up | command_auinotebook_tab_right_down
| command_auinotebook_tab_right_up | command_auinotebook_page_closed |
command_auinotebook_drag_done | command_auinotebook_bg_dclick
wxCalendar() = #wxCalendar{type=undefined | wxCalendarEventType()}
wxCalendarEventType() = calendar_sel_changed | calendar_day_changed |
calendar_month_changed | calendar_year_changed | calendar_doubleclicked |
calendar_weekday_clicked
```

```
wxChildFocus() = #wxChildFocus{type=undefined | wxChildFocusEventType()}
wxChildFocusEventType() = child_focus
wxClose() = #wxClose{type=undefined | wxCloseEventType()}
wxCloseEventType() = close_window | end_session | query_end_session
wxColourPicker() = #wxColourPicker{type=undefined |
wxColourPickerEventType(), colour=undefined | wx_colour() (see module wx)}
wxColourPickerEventType() = command_colourpicker_changed
wxCommand() = #wxCommand{type=undefined | wxCommandEventType(),
cmdString=undefined | chardata() (see module unicode), commandInt=undefined |
integer(), extraLong=undefined | integer()}
wxCommandEventType() = command_button_clicked | command_checkbox_clicked
| command_choice_selected | command_listbox_selected |
command_listbox_doubleclicked | command_text_updated | command_text_enter |
command_menu_selected | command_slider_updated | command_radiobox_selected
| command_radiobutton_selected | command_scrollbar_updated
| command_vlbox_selected | command_combobox_selected |
command_tool_rclicked | command_tool_enter | command_checklistbox_toggled
| command_togglebutton_clicked | command_left_click | command_left_dclick |
command_right_click | command_set_focus | command_kill_focus | command_enter
wxContextMenu() = #wxContextMenu{type=undefined | wxContextMenuEventType()}
wxContextMenuEventType() = context_menu
wxDate() = #wxDate{type=undefined | wxDateEventType(), date=undefined |
wx_datetime() (see module wx)}
wxDateEventType() = date_changed
wxDisplayChanged() = #wxDisplayChanged{type=undefined |
wxDisplayChangedEventType()}
wxDisplayChangedEventType() = display_changed
wxErase() = #wxErase{type=undefined | wxEraseEventType(), dc=undefined |
wxDC() (see module wxDC)}
wxEraseEventType() = erase_background
wxEventType() = wxAuiManagerEventType() | wxAuiNotebookEventType() |
wxCalendarEventType() | wxChildFocusEventType() | wxCloseEventType() |
wxColourPickerEventType() | wxCommandEventType() | wxContextMenuEventType()
| wxDateEventType() | wxDisplayChangedEventType() | wxEraseEventType() |
wxFileDirPickerEventType() | wxFocusEventType() | wxFontPickerEventType()
| wxGridEventType() | wxHelpEventType() | wxHtmlLinkEventType() |
wxIconizeEventType() | wxIdleEventType() | wxJoystickEventType()
| wxKeyEventType() | wxListEventType() | wxMaximizeEventType() |
wxMenuEventType() | wxMouseCaptureChangedEventType() | wxMouseEventType()
| wxMoveEventType() | wxNavigationKeyEventType() | wxNcPaintEventType() |
wxNotebookEventType() | wxPaintEventType() | wxPaletteChangedEventType()
| wxQueryNewPaletteEventType() | wxSashEventType() | wxScrollEventType()
| wxScrollWinEventType() | wxSetCursorEventType() | wxShowEventType()
| wxSizeEventType() | wxSpinEventType() | wxSplitterEventType()
| wxStyledTextEventType() | wxSysColourChangedEventType() |
wxTaskBarIconEventType() | wxTreeEventType() | wxUpdateUIEventType() |
wxWindowCreateEventType() | wxWindowDestroyEventType()
wxEvtHandler() = wx_object() (see module wx)
wxFileDirPicker() = #wxFileDirPicker{type=undefined |
wxFileDirPickerEventType(), path=undefined | chardata() (see module unicode)}
wxFileDirPickerEventType() = command_filepicker_changed |
command_dirpicker_changed
```

```
wxFocus() = #wxFocus{type=undefined | wxFocusEventType()}
wxFocusEventType() = set_focus | kill_focus
wxFontPicker() = #wxFontPicker{type=undefined | wxFontPickerEventType(),
font=undefined | wxFont() (see module wxFont)}
wxFontPickerEventType() = command_fontpicker_changed
wxGrid() = #wxGrid{type=undefined | wxGridEventType(), row=undefined |
integer(), col=undefined | integer(), x=undefined | integer(), y=undefined
| integer(), selecting=undefined | boolean(), control=undefined | boolean(),
meta=undefined | boolean(), shift=undefined | boolean(), alt=undefined |
boolean()}
wxGridEventType() = grid_cell_left_click | grid_cell_right_click |
grid_cell_left_dclick | grid_cell_right_dclick | grid_label_left_click |
grid_label_right_click | grid_label_left_dclick | grid_label_right_dclick
| grid_row_size | grid_col_size | grid_range_select | grid_cell_change
| grid_select_cell | grid_editor_shown | grid_editor_hidden |
grid_editor_created | grid_cell_begin_drag
wxHelp() = #wxHelp{type=undefined | wxHelpEventType()}
wxHelpEventType() = help | detailed_help
wxHtmlLink() = #wxHtmlLink{type=undefined | wxHtmlLinkEventType(),
linkInfo=undefined | wx_wxHtmlLinkInfo() (see module wx)}
wxHtmlLinkEventType() = command_html_link_clicked
wxIconize() = #wxIconize{type=undefined | wxIconizeEventType()}
wxIconizeEventType() = iconize
wxIdle() = #wxIdle{type=undefined | wxIdleEventType()}
wxIdleEventType() = idle
wxJoystick() = #wxJoystick{type=undefined | wxJoystickEventType()}
wxJoystickEventType() = joy_button_down | joy_button_up | joy_move |
joy_zmove
wxKey() = #wxKey{type=undefined | wxKeyEventType(), x=undefined | integer(),
y=undefined | integer(), keyCode=undefined | integer(), controlDown=undefined
| boolean(), shiftDown=undefined | boolean(), altDown=undefined
| boolean(), metaDown=undefined | boolean(), scanCode=undefined |
boolean(), uniChar=undefined | integer(), rawCode=undefined | integer(),
rawFlags=undefined | integer()}
wxKeyEventType() = char | char_hook | key_down | key_up
wxList() = #wxList{type=undefined | wxListEventType(), code=undefined
| integer(), oldItemIndex=undefined | integer(), itemIndex=undefined |
integer(), col=undefined | integer(), pointDrag=undefined | {X::integer(),
Y::integer()}}
wxListEventType() = command_list_begin_drag | command_list_begin_rdrag
| command_list_begin_label_edit | command_list_end_label_edit
| command_list_delete_item | command_list_delete_all_items |
command_list_key_down | command_list_insert_item | command_list_col_click
| command_list_col_right_click | command_list_col_begin_drag
| command_list_col_dragging | command_list_col_end_drag |
command_list_item_selected | command_list_item_deselected |
command_list_item_right_click | command_list_item_middle_click
| command_list_item_activated | command_list_item_focused |
command_list_cache_hint
wxMaximize() = #wxMaximize{type=undefined | wxMaximizeEventType()}
wxMaximizeEventType() = maximize
wxMenu() = #wxMenu{type=undefined | wxMenuEventType()}
```

```
wxMenuEventType() = menu_open | menu_close | menu_highlight
wxMouse() = #wxMouse{type=undefined | wxMouseEventType(), x=undefined
| integer(), y=undefined | integer(), leftDown=undefined | boolean(),
middleDown=undefined | boolean(), rightDown=undefined | boolean(),
controlDown=undefined | boolean(), shiftDown=undefined | boolean(),
altDown=undefined | boolean(), metaDown=undefined | boolean(),
wheelRotation=undefined | integer(), wheelDelta=undefined | integer(),
linesPerAction=undefined | integer()}
wxMouseCaptureChanged() = #wxMouseCaptureChanged{type=undefined |
wxMouseCaptureChangedEventType()}
wxMouseCaptureChangedEventType() = mouse_capture_changed
wxMouseEventType() = left_down | left_up | middle_down | middle_up |
right_down | right_up | motion | enter_window | leave_window | left_dclick
| middle_dclick | right_dclick | mousewheel | nc_left_down | nc_left_up
| nc_middle_down | nc_middle_up | nc_right_down | nc_right_up | nc_motion
| nc_enter_window | nc_leave_window | nc_left_dclick | nc_middle_dclick |
nc_right_dclick
wxMove() = #wxMove{type=undefined | wxMoveEventType()}
wxMoveEventType() = move
wxNavigationKey() = #wxNavigationKey{type=undefined |
wxNavigationKeyEventType(), flags=undefined | integer(), focus=undefined |
wxWindow() (see module wxWindow)}
wxNavigationKeyEventType() = navigation_key
wxNcPaint() = #wxNcPaint{type=undefined | wxNcPaintEventType()}
wxNcPaintEventType() = nc_paint
wxNotebook() = #wxNotebook{type=undefined | wxNotebookEventType()}
wxNotebookEventType() = command_notebook_page_changed |
command_notebook_page_changing
wxPaint() = #wxPaint{type=undefined | wxPaintEventType()}
wxPaintEventType() = paint | paint_icon
wxPaletteChanged() = #wxPaletteChanged{type=undefined |
wxPaletteChangedEventType()}
wxPaletteChangedEventType() = palette_changed
wxQueryNewPalette() = #wxQueryNewPalette{type=undefined |
wxQueryNewPaletteEventType()}
wxQueryNewPaletteEventType() = query_new_palette
wxSash() = #wxSash{type=undefined | wxSashEventType(), edge=undefined |
wx_enum() (see module wx), dragRect=undefined | {X::integer(), Y::integer(),
W::integer(), H::integer()}, dragStatus=undefined | wx_enum() (see module
wx)}
wxSashEventType() = sash_dragged
wxScroll() = #wxScroll{type=undefined | wxScrollEventType(),
commandInt=undefined | integer(), extraLong=undefined | integer()}
wxScrollEventType() = scroll_top | scroll_bottom | scroll_lineup |
scroll_linedown | scroll_pageup | scroll_pagedown | scroll_thumbtrack |
scroll_thumbrelease | scroll_changed
wxScrollWin() = #wxScrollWin{type=undefined | wxScrollWinEventType()}
wxScrollWinEventType() = scrollwin_top | scrollwin_bottom | scrollwin_lineup
| scrollwin_linedown | scrollwin_pageup | scrollwin_pagedown |
scrollwin_thumbtrack | scrollwin_thumbrelease
wxSetCursor() = #wxSetCursor{type=undefined | wxSetCursorEventType()}
wxSetCursorEventType() = set_cursor
```

```
wxShow() = #wxShow{type=undefined | wxShowEventType()}
wxShowEventType() = show
wxSize() = #wxSize{type=undefined | wxSizeEventType(), size=undefined |
{W::integer(), H::integer()}, rect=undefined | {X::integer(), Y::integer(),
W::integer(), H::integer()}}
wxSizeEventType() = size
wxSpin() = #wxSpin{type=undefined | wxSpinEventType(), commandInt=undefined |
integer()}
wxSpinEventType() = command_spinctrl_updated | spin_up | spin_down | spin
wxSplitter() = #wxSplitter{type=undefined | wxSplitterEventType()}
wxSplitterEventType() = command_splitter_sash_pos_changed |
command_splitter_sash_pos_changing | command_splitter_doubleclicked |
command_splitter_unsplit
wxStyledText() = #wxStyledText{type=undefined | wxStyledTextEventType(),
position=undefined | integer(), key=undefined | integer(),
modifiers=undefined | integer(), modificationType=undefined | integer(),
text=undefined | chardata() (see module unicode), length=undefined |
integer(), linesAdded=undefined | integer(), line=undefined | integer(),
foldLevelNow=undefined | integer(), foldLevelPrev=undefined | integer(),
margin=undefined | integer(), message=undefined | integer(), wParam=undefined
| integer(), lParam=undefined | integer(), listType=undefined | integer(),
x=undefined | integer(), y=undefined | integer(), dragText=undefined |
chardata() (see module unicode), dragAllowMove=undefined | boolean(),
dragResult=undefined | wx_enum() (see module wx)}
wxStyledTextEventType() = stc_change | stc_styleneeded | stc_charadded |
stc_savepointreached | stc_savepointleft | stc_romodifyattempt | stc_key
| stc_doubleclick | stc_updateui | stc_modified | stc_macrorecord |
stc_marginclick | stc_needshown | stc_painted | stc_userlistselection
| stc_uridropped | stc_dwellstart | stc_dwellend | stc_start_drag
| stc_drag_over | stc_do_drop | stc_zoom | stc_hotspot_click |
stc_hotspot_dclick | stc_calltip_click | stc_autocomp_selection
wxSysColourChanged() = #wxSysColourChanged{type=undefined |
wxSysColourChangedEventType()}
wxSysColourChangedEventType() = sys_colour_changed
wxTaskBarIcon() = #wxTaskBarIcon{type=undefined | wxTaskBarIconEventType()}
wxTaskBarIconEventType() = taskbar_move | taskbar_left_down | taskbar_left_up
| taskbar_right_down | taskbar_right_up | taskbar_left_dclick |
taskbar_right_dclick
wxTree() = #wxTree{type=undefined | wxTreeEventType(), item=undefined
| integer(), itemOld=undefined | integer(), pointDrag=undefined |
{X::integer(), Y::integer()}}
wxTreeEventType() = command_tree_begin_drag | command_tree_begin_rdrag
| command_tree_begin_label_edit | command_tree_end_label_edit |
command_tree_delete_item | command_tree_get_info | command_tree_set_info
| command_tree_item_expanded | command_tree_item_expanding |
command_tree_item_collapsed | command_tree_item_collapsing |
command_tree_sel_changed | command_tree_sel_changing | command_tree_key_down
| command_tree_item_activated | command_tree_item_right_click
| command_tree_item_middle_click | command_tree_end_drag |
command_tree_state_image_click | command_tree_item_gettooltip |
command_tree_item_menu
wxUpdateUI() = #wxUpdateUI{type=undefined | wxUpdateUIEventType()}
```

```
wxUpdateUIEventType() = update_ui
wxWindowCreate() = #wxWindowCreate{type=undefined |
wxWindowCreateEventType()}
wxWindowCreateEventType() = create
wxWindowDestroy() = #wxWindowDestroy{type=undefined |
wxWindowDestroyEventType()}
wxWindowDestroyEventType() = destroy
```

Exports

connect(This::wxEvtHandler(), EventType::wxEventType()) -> ok

Equivalent to *connect(This, EventType, [])*

connect(This::wxEvtHandler(), EventType::wxEventType(), Option::[Option]) -> ok

Types:

Option = {id, integer()} | {lastId, integer()} | {skip, boolean()} | {callback, function()} | {userData, term()}

This function subscribes the to events of EventType, in the range id, lastId. The events will be received as messages if no callback is supplied.

Options: {id, integer()}, The identifier (or first of the identifier range) to be associated with this event handler. Default ?wxID_ANY {lastId, integer()}, The second part of the identifier range. If used 'id' must be set as the starting identifier range. Default ?wxID_ANY {skip, boolean()}, If skip is true further event_handlers will be called. This is not used if the 'callback' option is used. Default false. {callback, function()} Use a callback fun(EventRecord::wx(), EventObject::wxObject()) to process the event. Default not specfied i.e. a message will be delivered to the process calling this function. {userData, term()} An erlang term that will be sent with the event. Default: [].

disconnect(This::wxEvtHandler()) -> boolean()

Equivalent to *disconnect(This, null, [])* Can also have an optional callback Fun() as an additional last argument.

disconnect(This::wxEvtHandler(), EventType::wxEventType()) -> boolean()

Equivalent to *disconnect(This, EventType, [])*

disconnect(This::wxEvtHandler(), EventType::wxEventType(), Option::[Option]) -> boolean()

Types:

Option = {id, integer()} | {lastId, integer()} | {callback, function()}

See **external documentation** This function unsubscribes the process or callback fun from the event handler. EventType may be the atom 'null' to match any eventtype. Notice that the options skip and userdata is not used to match the eventhandler.

wxFileDataObject

Erlang module

See external documentation: **wxFileDataObject**.

This class is derived (and can use functions) from:
wxDataObject

DATA TYPES

`wxFileDataObject()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFileDataObject()`

See **external documentation**.

`addFile(This, Filename) -> ok`

Types:

`This = wxFileDataObject()`
`Filename = chardata() (see module unicode)`

See **external documentation**.

`getFilenames(This) -> [charlist() (see module unicode)]`

Types:

`This = wxFileDataObject()`

See **external documentation**.

`destroy(This::wxFileDataObject()) -> ok`

Destroys this object, do not use object again

wxFileDialog

Erlang module

See external documentation: **wxFileDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxFileDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent) -> wxFileDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxFileDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`

`Option = {message, chardata() (see module unicode)} | {defaultDir, chardata() (see module unicode)} | {defaultFile, chardata() (see module unicode)} | {wildCard, chardata() (see module unicode)} | {style, integer()} | {pos, {X::integer(), Y::integer()}} | {sz, {W::integer(), H::integer()}}`

See **external documentation**.

`getDirectory(This) -> charlist() (see module unicode)`

Types:

`This = wxFileDialog()`

See **external documentation**.

`getFilename(This) -> charlist() (see module unicode)`

Types:

`This = wxFileDialog()`

See **external documentation**.

getFilenames(This) -> [charlist() (see module unicode)]

Types:

This = wxFileDialog()

See **external documentation**.

getFilterIndex(This) -> integer()

Types:

This = wxFileDialog()

See **external documentation**.

getMessage(This) -> charlist() (see module unicode)

Types:

This = wxFileDialog()

See **external documentation**.

getPath(This) -> charlist() (see module unicode)

Types:

This = wxFileDialog()

See **external documentation**.

getPaths(This) -> [charlist() (see module unicode)]

Types:

This = wxFileDialog()

See **external documentation**.

getWildcard(This) -> charlist() (see module unicode)

Types:

This = wxFileDialog()

See **external documentation**.

setDirectory(This, Dir) -> ok

Types:

This = wxFileDialog()

Dir = chardata() (see module unicode)

See **external documentation**.

setFilename(This, Name) -> ok

Types:

This = wxFileDialog()

Name = chardata() (see module unicode)

See **external documentation**.

setFilterIndex(This, FilterIndex) -> ok

Types:

This = wxFileDialog()
FilterIndex = integer()

See **external documentation**.

setMessage(This, Message) -> ok

Types:

This = wxFileDialog()
Message = chardata() (see module unicode)

See **external documentation**.

setPath(This, Path) -> ok

Types:

This = wxFileDialog()
Path = chardata() (see module unicode)

See **external documentation**.

setWildcard(This, WildCard) -> ok

Types:

This = wxFileDialog()
WildCard = chardata() (see module unicode)

See **external documentation**.

destroy(This::wxFileDialog()) -> ok

Destroys this object, do not use object again

wxFileDirPickerEvent

Erlang module

See external documentation: **wxFileDirPickerEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_filepicker_changed, *command_dirpicker_changed*

See also the message variant *#wxFileDirPicker{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxFileDirPickerEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getPath(This) -> charlist() (see module unicode)`

Types:

`This = wxFileDirPickerEvent()`

See **external documentation**.

wxFilePickerCtrl

Erlang module

See external documentation: **wxFilePickerCtrl**.

This class is derived (and can use functions) from:
wxPickerBase
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxFilePickerCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFilePickerCtrl()`

See **external documentation**.

`new(Parent, Id) -> wxFilePickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxFilePickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {path, chardata() (see module unicode)} | {message, chardata()
(see module unicode)} | {wildcard, chardata() (see module unicode)} |
{pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}}
| {style, integer()} | {validator, wx_object() (see module wx)}
```

See **external documentation**.

`create(This, Parent, Id) -> boolean()`

Types:

```
This = wxFilePickerCtrl()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

```
create(This, Parent, Id, Option::[Option]) -> boolean()
```

Types:

```
This = wxFilePickerCtrl()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {path, chardata() (see module unicode)} | {message, chardata()
(see module unicode)} | {wildcard, chardata() (see module unicode)} |
{pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}}
| {style, integer()} | {validator, wx_object() (see module wx)}
```

See **external documentation**.

```
getPath(This) -> charlist() (see module unicode)
```

Types:

```
This = wxFilePickerCtrl()
```

See **external documentation**.

```
setPath(This, Str) -> ok
```

Types:

```
This = wxFilePickerCtrl()
Str = chardata() (see module unicode)
```

See **external documentation**.

```
destroy(This::wxFilePickerCtrl()) -> ok
```

Destroys this object, do not use object again

wxFindReplaceData

Erlang module

See external documentation: **wxFindReplaceData**.

DATA TYPES

`wxFindReplaceData()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFindReplaceData()`

See **external documentation**.

`new(Flags) -> wxFindReplaceData()`

Types:

`Flags = integer()`

See **external documentation**.

`getFindString(This) -> charlist() (see module unicode)`

Types:

`This = wxFindReplaceData()`

See **external documentation**.

`getReplaceString(This) -> charlist() (see module unicode)`

Types:

`This = wxFindReplaceData()`

See **external documentation**.

`getFlags(This) -> integer()`

Types:

`This = wxFindReplaceData()`

See **external documentation**.

`setFlags(This, Flags) -> ok`

Types:

`This = wxFindReplaceData()`

`Flags = integer()`

See **external documentation**.

setFindString(This, Str) -> ok

Types:

```
This = wxFindReplaceData()
Str = chardata() (see module unicode)
```

See **external documentation**.

setReplaceString(This, Str) -> ok

Types:

```
This = wxFindReplaceData()
Str = chardata() (see module unicode)
```

See **external documentation**.

destroy(This::wxFindReplaceData()) -> ok

Destroys this object, do not use object again

wxFindReplaceDialog

Erlang module

See external documentation: **wxFindReplaceDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxFindReplaceDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFindReplaceDialog()`

See **external documentation**.

`new(Parent, Data, Title) -> wxFindReplaceDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Data = wxFindReplaceData() (see module wxFindReplaceData)
Title = chardata() (see module unicode)
```

Equivalent to *new(Parent, Data, Title, [])*.

`new(Parent, Data, Title, Option::[Option]) -> wxFindReplaceDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Data = wxFindReplaceData() (see module wxFindReplaceData)
Title = chardata() (see module unicode)
Option = {style, integer()}
```

See **external documentation**.

`create(This, Parent, Data, Title) -> boolean()`

Types:

```
This = wxFindReplaceDialog()
Parent = wxWindow() (see module wxWindow)
Data = wxFindReplaceData() (see module wxFindReplaceData)
Title = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Data, Title, [])*.

create(This, Parent, Data, Title, Option::[Option]) -> boolean()

Types:

```
This = wxFindReplaceDialog()
Parent = wxWindow() (see module wxWindow)
Data = wxFindReplaceData() (see module wxFindReplaceData)
Title = chardata() (see module unicode)
Option = {style, integer()}
```

See **external documentation**.

getData(This) -> wxFindReplaceData() (see module wxFindReplaceData)

Types:

```
This = wxFindReplaceDialog()
```

See **external documentation**.

destroy(This::wxFindReplaceDialog()) -> ok

Destroys this object, do not use object again

wxFlexGridSizer

Erlang module

See external documentation: **wxFlexGridSizer**.

This class is derived (and can use functions) from:
wxGridSizer
wxSizer

DATA TYPES

`wxFlexGridSizer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Cols) -> wxFlexGridSizer()`

Types:

`Cols = integer()`

Equivalent to *new(Cols, [])*.

`new(Cols, Option::[Option]) -> wxFlexGridSizer()`

Types:

`Cols = integer()`
`Option = {vgap, integer()} | {hgap, integer()}`

See **external documentation**.

`new(Rows, Cols, Vgap, Hgap) -> wxFlexGridSizer()`

Types:

`Rows = integer()`
`Cols = integer()`
`Vgap = integer()`
`Hgap = integer()`

See **external documentation**.

`addGrowableCol(This, Idx) -> ok`

Types:

`This = wxFlexGridSizer()`
`Idx = integer()`

Equivalent to *addGrowableCol(This, Idx, [])*.

`addGrowableCol(This, Idx, Option::[Option]) -> ok`

Types:

```
This = wxFlexGridSizer()
Idx = integer()
Option = {proportion, integer()}
```

See **external documentation**.

addGrowableRow(This, Idx) -> ok

Types:

```
This = wxFlexGridSizer()
Idx = integer()
```

Equivalent to *addGrowableRow(This, Idx, [])*.

addGrowableRow(This, Idx, Option::[Option]) -> ok

Types:

```
This = wxFlexGridSizer()
Idx = integer()
Option = {proportion, integer()}
```

See **external documentation**.

getFlexibleDirection(This) -> integer()

Types:

```
This = wxFlexGridSizer()
```

See **external documentation**.

getNonFlexibleGrowMode(This) -> wx_enum() (see module wx)

Types:

```
This = wxFlexGridSizer()
```

See **external documentation**.
Res = ?wxFLEX_GROWMODE_NONE | ?wxFLEX_GROWMODE_SPECIFIED | ?wxFLEX_GROWMODE_ALL

removeGrowableCol(This, Idx) -> ok

Types:

```
This = wxFlexGridSizer()
Idx = integer()
```

See **external documentation**.

removeGrowableRow(This, Idx) -> ok

Types:

```
This = wxFlexGridSizer()
Idx = integer()
```

See **external documentation**.

setFlexibleDirection(This, Direction) -> ok

Types:

```
This = wxFlexGridSizer()
Direction = integer()
```

See **external documentation**.

setNonFlexibleGrowMode(This, Mode) -> ok

Types:

```
This = wxFlexGridSizer()
Mode = wx_enum() (see module wx)
```

See **external documentation**.
Mode = ?wxFLEX_GROWMODE_NONE | ?wxFLEX_GROWMODE_SPECIFIED | ?wxFLEX_GROWMODE_ALL

destroy(This::wxFlexGridSizer()) -> ok

Destroys this object, do not use object again

wxFocusEvent

Erlang module

See external documentation: **wxFocusEvent**.

Use *wxEvtHandler:connect/3* with EventType:

set_focus, *kill_focus*

See also the message variant *#wxFocus{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxFocusEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getWindow(This) -> wxWindow() (see module wxWindow)`

Types:

`This = wxFocusEvent()`

See **external documentation**.

wxFont

Erlang module

See external documentation: **wxFont**.

DATA TYPES

`wxFont()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFont()`

See **external documentation**.

`new(Fontname) -> wxFont()`

Types:

`Fontname = chardata() (see module unicode)`

See **external documentation**.

`new(Size, Family, Style, Weight) -> wxFont()`

Types:

`Size = integer()`
`Family = wx_enum() (see module wx)`
`Style = wx_enum() (see module wx)`
`Weight = integer()`

Equivalent to *new(Size, Family, Style, Weight, [])*.

`new(Size, Family, Style, Weight, Option::[Option]) -> wxFont()`

Types:

`Size = integer()`
`Family = wx_enum() (see module wx)`
`Style = wx_enum() (see module wx)`
`Weight = integer()`
`Option = {underlined, boolean()} | {face, chardata() (see module unicode)} | {encoding, wx_enum() (see module wx)}`

See **external documentation**.

Encoding = ?wxFONTENCODING_SYSTEM | ?wxFONTENCODING_DEFAULT | ?wxFONTENCODING_ISO8859_1 | ?wxFONTENCODING_ISO8859_2 | ?wxFONTENCODING_ISO8859_3 | ?wxFONTENCODING_ISO8859_4 | ?wxFONTENCODING_ISO8859_5 | ?wxFONTENCODING_ISO8859_6 | ?wxFONTENCODING_ISO8859_7 | ?wxFONTENCODING_ISO8859_8 | ?wxFONTENCODING_ISO8859_9 | ?wxFONTENCODING_ISO8859_10 | ?wxFONTENCODING_ISO8859_11 | ?wxFONTENCODING_ISO8859_12 | ?wxFONTENCODING_ISO8859_13 | ?wxFONTENCODING_ISO8859_14 | ?wxFONTENCODING_ISO8859_15

| ?wxFONTENCODING_ISO8859_MAX | ?wxFONTENCODING_KOI8 | ?wxFONTENCODING_KOI8_U | ?wxFONTENCODING_ALTERNATIVE | ?wxFONTENCODING_BULGARIAN | ?wxFONTENCODING_CP437 | ?wxFONTENCODING_CP850 | ?wxFONTENCODING_CP852 | ?wxFONTENCODING_CP855 | ?wxFONTENCODING_CP866 | ?wxFONTENCODING_CP874 | ?wxFONTENCODING_CP932 | ?wxFONTENCODING_CP936 | ?wxFONTENCODING_CP949 | ?wxFONTENCODING_CP950 | ?wxFONTENCODING_CP1250 | ?wxFONTENCODING_CP1251 | ?wxFONTENCODING_CP1252 | ?wxFONTENCODING_CP1253 | ?wxFONTENCODING_CP1254 | ?wxFONTENCODING_CP1255 | ?wxFONTENCODING_CP1256 | ?wxFONTENCODING_CP1257 | ?wxFONTENCODING_CP12_MAX | ?wxFONTENCODING_UTF7 | ?wxFONTENCODING_UTF8 | ?wxFONTENCODING_EUC_JP | ?wxFONTENCODING_UTF16BE | ?wxFONTENCODING_UTF16LE | ?wxFONTENCODING_UTF32BE | ?wxFONTENCODING_UTF32LE | ?wxFONTENCODING_MACROMAN | ?wxFONTENCODING_MACJAPANESE | ?wxFONTENCODING_MACCHINESETRAD | ?wxFONTENCODING_MACKOREAN | ?wxFONTENCODING_MACARABIC | ?wxFONTENCODING_MACHEBREW | ?wxFONTENCODING_MACGREEK | ?wxFONTENCODING_MACCYRILLIC | ?wxFONTENCODING_MACDEVANAGARI | ?wxFONTENCODING_MACGURMUKHI | ?wxFONTENCODING_MACGUJARATI | ?wxFONTENCODING_MACORIYA | ?wxFONTENCODING_MACBENGALI | ?wxFONTENCODING_MACTAMIL | ?wxFONTENCODING_MACTELUGU | ?wxFONTENCODING_MACKANNADA | ?wxFONTENCODING_MACMALAJALAM | ?wxFONTENCODING_MACSINHALESE | ?wxFONTENCODING_MACBURMESE | ?wxFONTENCODING_MACKHMER | ?wxFONTENCODING_MACTHAI | ?wxFONTENCODING_MACLAOTIAN | ?wxFONTENCODING_MACGEORGIAN | ?wxFONTENCODING_MACARMENIAN | ?wxFONTENCODING_MACCHINESESIMP | ?wxFONTENCODING_MACTIBETAN | ?wxFONTENCODING_MACMONGOLIAN | ?wxFONTENCODING_MACETHIOPIC | ?wxFONTENCODING_MACCENTRALEUR | ?wxFONTENCODING_MACVIATNAMESE | ?wxFONTENCODING_MACARABICEXT | ?wxFONTENCODING_MACSYMBOL | ?wxFONTENCODING_MACDINGBATS | ?wxFONTENCODING_MACTURKISH | ?wxFONTENCODING_MACCROATIAN | ?wxFONTENCODING_MACICELANDIC | ?wxFONTENCODING_MACROMANIAN | ?wxFONTENCODING_MACCELTIC | ?wxFONTENCODING_MACGAELIC | ?wxFONTENCODING_MACKEYBOARD | ?wxFONTENCODING_MAX | ?wxFONTENCODING_MACMIN | ?wxFONTENCODING_MACMAX | ?wxFONTENCODING_UTF16 | ?wxFONTENCODING_UTF32 | ?wxFONTENCODING_UNICODE | ?wxFONTENCODING_GB2312 | ?wxFONTENCODING_BIG5 | ?wxFONTENCODING_SHIFT_JIS
Family = ?wxFONTFAMILY_DEFAULT | ?wxFONTFAMILY_DECORATIVE | ?wxFONTFAMILY_ROMAN | ?wxFONTFAMILY_SCRIPT | ?wxFONTFAMILY_SWISS | ?wxFONTFAMILY_MODERN | ?wxFONTFAMILY_TELETYPE | ?wxFONTFAMILY_MAX | ?wxFONTFAMILY_UNKNOWN
Style = ?wxFONTSTYLE_NORMAL | ?wxFONTSTYLE_ITALIC | ?wxFONTSTYLE_SLANT | ?wxFONTSTYLE_MAX

`isFixedWidth(This) -> boolean()`

Types:

```
This = wxFont()
```

See **external documentation**.

`getDefaultEncoding() -> wx_enum() (see module wx)`

See **external documentation**.
Res = ?wxFONTENCODING_SYSTEM | ?wxFONTENCODING_DEFAULT | ?wxFONTENCODING_ISO8859_1 | ?wxFONTENCODING_ISO8859_2 | ?wxFONTENCODING_ISO8859_3 | ?wxFONTENCODING_ISO8859_4 | ?wxFONTENCODING_ISO8859_5 | ?wxFONTENCODING_ISO8859_6 | ?

wxFONTENCODING_ISO8859_7 | ?wxFONTENCODING_ISO8859_8 | ?wxFONTENCODING_ISO8859_9 | ?wxFONTENCODING_ISO8859_10 | ?wxFONTENCODING_ISO8859_11 | ?wxFONTENCODING_ISO8859_12 | ?wxFONTENCODING_ISO8859_13 | ?wxFONTENCODING_ISO8859_14 | ?wxFONTENCODING_ISO8859_15 | ?wxFONTENCODING_ISO8859_MAX | ?wxFONTENCODING_KOI8 | ?wxFONTENCODING_KOI8_U | ?wxFONTENCODING_ALTERNATIVE | ?wxFONTENCODING_BULGARIAN | ?wxFONTENCODING_CP437 | ?wxFONTENCODING_CP850 | ?wxFONTENCODING_CP852 | ?wxFONTENCODING_CP855 | ?wxFONTENCODING_CP866 | ?wxFONTENCODING_CP874 | ?wxFONTENCODING_CP932 | ?wxFONTENCODING_CP936 | ?wxFONTENCODING_CP949 | ?wxFONTENCODING_CP950 | ?wxFONTENCODING_CP1250 | ?wxFONTENCODING_CP1251 | ?wxFONTENCODING_CP1252 | ?wxFONTENCODING_CP1253 | ?wxFONTENCODING_CP1254 | ?wxFONTENCODING_CP1255 | ?wxFONTENCODING_CP1256 | ?wxFONTENCODING_CP1257 | ?wxFONTENCODING_CP12_MAX | ?wxFONTENCODING_UTF7 | ?wxFONTENCODING_UTF8 | ?wxFONTENCODING_EUC_JP | ?wxFONTENCODING_UTF16BE | ?wxFONTENCODING_UTF16LE | ?wxFONTENCODING_UTF32BE | ?wxFONTENCODING_UTF32LE | ?wxFONTENCODING_MACROMAN | ?wxFONTENCODING_MACJAPANESE | ?wxFONTENCODING_MACCHINESETRAD | ?wxFONTENCODING_MACKOREAN | ?wxFONTENCODING_MACARABIC | ?wxFONTENCODING_MACHEBREW | ?wxFONTENCODING_MACGREEK | ?wxFONTENCODING_MACCYRILLIC | ?wxFONTENCODING_MACDEVANAGARI | ?wxFONTENCODING_MACGURMUKHI | ?wxFONTENCODING_MACGUJARATI | ?wxFONTENCODING_MACORIYA | ?wxFONTENCODING_MACBENGALI | ?wxFONTENCODING_MACTAMIL | ?wxFONTENCODING_MACTELUGU | ?wxFONTENCODING_MACKANNADA | ?wxFONTENCODING_MACMALAJALAM | ?wxFONTENCODING_MACSINHALESE | ?wxFONTENCODING_MACBURMESE | ?wxFONTENCODING_MACKHMER | ?wxFONTENCODING_MACTHAI | ?wxFONTENCODING_MACLAOTIAN | ?wxFONTENCODING_MACGEORGIAN | ?wxFONTENCODING_MACARMENIAN | ?wxFONTENCODING_MACCHINESESIMP | ?wxFONTENCODING_MACTIBETAN | ?wxFONTENCODING_MACMONGOLIAN | ?wxFONTENCODING_MACETHIOPIC | ?wxFONTENCODING_MACCENTRALEUR | ?wxFONTENCODING_MACVIATNAMESE | ?wxFONTENCODING_MACARABICEXT | ?wxFONTENCODING_MACSYMBOL | ?wxFONTENCODING_MACDINGBATS | ?wxFONTENCODING_MACTURKISH | ?wxFONTENCODING_MACCROATIAN | ?wxFONTENCODING_MACICELANDIC | ?wxFONTENCODING_MACROMANIAN | ?wxFONTENCODING_MACCELTIC | ?wxFONTENCODING_MACGAELIC | ?wxFONTENCODING_MACKEYBOARD | ?wxFONTENCODING_MAX | ?wxFONTENCODING_MACMIN | ?wxFONTENCODING_MACMAX | ?wxFONTENCODING_UTF16 | ?wxFONTENCODING_UTF32 | ?wxFONTENCODING_UNICODE | ?wxFONTENCODING_GB2312 | ?wxFONTENCODING_BIG5 | ?wxFONTENCODING_SHIFT_JIS

`getFaceName(This) -> charlist() (see module unicode)`

Types:

`This = wxFont()`

See **external documentation**.

`getFamily(This) -> wx_enum() (see module wx)`

Types:

`This = wxFont()`

See **external documentation**.
Res = ?wxFONTFAMILY_DEFAULT | ?wxFONTFAMILY_DECORATIVE | ?wxFONTFAMILY_ROMAN | ?wxFONTFAMILY_SCRIPT | ?wxFONTFAMILY_SWISS | ?wxFONTFAMILY_MODERN | ?wxFONTFAMILY_TELETYPE | ?wxFONTFAMILY_MAX | ?wxFONTFAMILY_UNKNOWN

getNativeFontInfoDesc(This) -> charlist() (see module unicode)

Types:

This = wxFont()

See **external documentation**.

getNativeFontInfoUserDesc(This) -> charlist() (see module unicode)

Types:

This = wxFont()

See **external documentation**.

getPointSize(This) -> integer()

Types:

This = wxFont()

See **external documentation**.

getStyle(This) -> wx_enum() (see module wx)

Types:

This = wxFont()

See **external documentation**.
Res = ?wxFONTSTYLE_NORMAL | ?wxFONTSTYLE_ITALIC | ?wxFONTSTYLE_SLANT | ?wxFONTSTYLE_MAX

getUnderlined(This) -> boolean()

Types:

This = wxFont()

See **external documentation**.

getWeight(This) -> integer()

Types:

This = wxFont()

See **external documentation**.

ok(This) -> boolean()

Types:

This = wxFont()

See **external documentation**.

setDefaultEncoding(Encoding) -> ok

Types:

Encoding = wx_enum() (see module wx)

See **external documentation**.
Encoding = ?wxFONTENCODING_SYSTEM | ?wxFONTENCODING_DEFAULT | ?wxFONTENCODING_ISO8859_1 | ?wxFONTENCODING_ISO8859_2 | ?wxFONTENCODING_ISO8859_3 | ?wxFONTENCODING_ISO8859_4 | ?wxFONTENCODING_ISO8859_5 | ?wxFONTENCODING_ISO8859_6 | ?

wxFONTENCODING_ISO8859_7 | ?wxFONTENCODING_ISO8859_8 | ?wxFONTENCODING_ISO8859_9 | ?wxFONTENCODING_ISO8859_10 | ?wxFONTENCODING_ISO8859_11 | ?wxFONTENCODING_ISO8859_12 | ?wxFONTENCODING_ISO8859_13 | ?wxFONTENCODING_ISO8859_14 | ?wxFONTENCODING_ISO8859_15 | ?wxFONTENCODING_ISO8859_MAX | ?wxFONTENCODING_KOI8 | ?wxFONTENCODING_KOI8_U | ?wxFONTENCODING_ALTERNATIVE | ?wxFONTENCODING_BULGARIAN | ?wxFONTENCODING_CP437 | ?wxFONTENCODING_CP850 | ?wxFONTENCODING_CP852 | ?wxFONTENCODING_CP855 | ?wxFONTENCODING_CP866 | ?wxFONTENCODING_CP874 | ?wxFONTENCODING_CP932 | ?wxFONTENCODING_CP936 | ?wxFONTENCODING_CP949 | ?wxFONTENCODING_CP950 | ?wxFONTENCODING_CP1250 | ?wxFONTENCODING_CP1251 | ?wxFONTENCODING_CP1252 | ?wxFONTENCODING_CP1253 | ?wxFONTENCODING_CP1254 | ?wxFONTENCODING_CP1255 | ?wxFONTENCODING_CP1256 | ?wxFONTENCODING_CP1257 | ?wxFONTENCODING_CP12_MAX | ?wxFONTENCODING_UTF7 | ?wxFONTENCODING_UTF8 | ?wxFONTENCODING_EUC_JP | ?wxFONTENCODING_UTF16BE | ?wxFONTENCODING_UTF16LE | ?wxFONTENCODING_UTF32BE | ?wxFONTENCODING_UTF32LE | ?wxFONTENCODING_MACROMAN | ?wxFONTENCODING_MACJAPANESE | ?wxFONTENCODING_MACCHINESETRAD | ?wxFONTENCODING_MACKOREAN | ?wxFONTENCODING_MACARABIC | ?wxFONTENCODING_MACHEBREW | ?wxFONTENCODING_MACGREEK | ?wxFONTENCODING_MACCYRILLIC | ?wxFONTENCODING_MACDEVANAGARI | ?wxFONTENCODING_MACGURMUKHI | ?wxFONTENCODING_MACGUJARATI | ?wxFONTENCODING_MACORIYA | ?wxFONTENCODING_MACBENGALI | ?wxFONTENCODING_MACTAMIL | ?wxFONTENCODING_MACTELUGU | ?wxFONTENCODING_MACKANNADA | ?wxFONTENCODING_MACMALAJALAM | ?wxFONTENCODING_MACSINHALESE | ?wxFONTENCODING_MACBURMESE | ?wxFONTENCODING_MACKHMER | ?wxFONTENCODING_MACTHAI | ?wxFONTENCODING_MACLAOTIAN | ?wxFONTENCODING_MACGEORGIAN | ?wxFONTENCODING_MACARMENIAN | ?wxFONTENCODING_MACCHINESESIMP | ?wxFONTENCODING_MACTIBETAN | ?wxFONTENCODING_MACMONGOLIAN | ?wxFONTENCODING_MACETHIOPIC | ?wxFONTENCODING_MACCENTRALEUR | ?wxFONTENCODING_MACVIATNAMESE | ?wxFONTENCODING_MACARABICEXT | ?wxFONTENCODING_MACSYMBOL | ?wxFONTENCODING_MACDINGBATS | ?wxFONTENCODING_MACTURKISH | ?wxFONTENCODING_MACCROATIAN | ?wxFONTENCODING_MACICELANDIC | ?wxFONTENCODING_MACROMANIAN | ?wxFONTENCODING_MACCELTIC | ?wxFONTENCODING_MACGAELIC | ?wxFONTENCODING_MACKEYBOARD | ?wxFONTENCODING_MAX | ?wxFONTENCODING_MACMIN | ?wxFONTENCODING_MACMAX | ?wxFONTENCODING_UTF16 | ?wxFONTENCODING_UTF32 | ?wxFONTENCODING_UNICODE | ?wxFONTENCODING_GB2312 | ?wxFONTENCODING_BIG5 | ?wxFONTENCODING_SHIFT_JIS

`setFaceName(This, FaceName) -> boolean()`

Types:

`This = wxFont()`

`FaceName = chardata() (see module unicode)`

See **external documentation**.

`setFamily(This, Family) -> ok`

Types:

`This = wxFont()`

`Family = wx_enum() (see module wx)`

See **external documentation**.

Family = ?wxFONTFAMILY_DEFAULT | ?wxFONTFAMILY_DECORATIVE | ?wxFONTFAMILY_ROMAN | ?wxFONTFAMILY_SCRIPT | ?wxFONTFAMILY_SWISS | ?wxFONTFAMILY_MODERN | ?wxFONTFAMILY_TELETYPE | ?wxFONTFAMILY_MAX | ?wxFONTFAMILY_UNKNOWN

`setPointSize(This, PointSize) -> ok`

Types:

```
This = wxFont()
PointSize = integer()
```

See **external documentation**.

`setStyle(This, Style) -> ok`

Types:

```
This = wxFont()
Style = wx_enum() (see module wx)
```

See **external documentation**.
Style = ?wxFONTSTYLE_NORMAL | ?wxFONTSTYLE_ITALIC | ?wxFONTSTYLE_SLANT | ?wxFONTSTYLE_MAX

`setUnderlined(This, Underlined) -> ok`

Types:

```
This = wxFont()
Underlined = boolean()
```

See **external documentation**.

`setWeight(This, Weight) -> ok`

Types:

```
This = wxFont()
Weight = integer()
```

See **external documentation**.

`destroy(This::wxFont()) -> ok`

Destroys this object, do not use object again

wxFontData

Erlang module

See external documentation: **wxFontData**.

DATA TYPES

`wxFontData()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFontData()`

See **external documentation**.

`new(Data) -> wxFontData()`

Types:

`Data = wxFontData()`

See **external documentation**.

`enableEffects(This, Flag) -> ok`

Types:

`This = wxFontData()`
`Flag = boolean()`

See **external documentation**.

`getAllowSymbols(This) -> boolean()`

Types:

`This = wxFontData()`

See **external documentation**.

`getColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxFontData()`

See **external documentation**.

`getChosenFont(This) -> wxFont() (see module wxFont)`

Types:

`This = wxFontData()`

See **external documentation**.

getEnableEffects(This) -> boolean()

Types:

This = wxFontData()

See **external documentation**.

getInitialFont(This) -> wxFont() (see module wxFont)

Types:

This = wxFontData()

See **external documentation**.

getShowHelp(This) -> boolean()

Types:

This = wxFontData()

See **external documentation**.

setAllowSymbols(This, Flag) -> ok

Types:

This = wxFontData()

Flag = boolean()

See **external documentation**.

setChosenFont(This, Font) -> ok

Types:

This = wxFontData()

Font = wxFont() (see module wxFont)

See **external documentation**.

setColour(This, Colour) -> ok

Types:

This = wxFontData()

Colour = wx_colour() (see module wx)

See **external documentation**.

setInitialFont(This, Font) -> ok

Types:

This = wxFontData()

Font = wxFont() (see module wxFont)

See **external documentation**.

setRange(This, MinRange, MaxRange) -> ok

Types:

This = wxFontData()

MinRange = integer()

```
MaxRange = integer()
```

See **external documentation**.

setShowHelp(This, Flag) -> ok

Types:

```
This = wxFontData()
Flag = boolean()
```

See **external documentation**.

destroy(This::wxFontData()) -> ok

Destroys this object, do not use object again

wxFontDialog

Erlang module

See external documentation: **wxFontDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxFontDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFontDialog()`

See **external documentation**.

`new(Parent, Data) -> wxFontDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Data = wxFontData() (see module wxFontData)
```

See **external documentation**.

`create(This, Parent, Data) -> boolean()`

Types:

```
This = wxFontDialog()
Parent = wxWindow() (see module wxWindow)
Data = wxFontData() (see module wxFontData)
```

See **external documentation**.

`getFontData(This) -> wxFontData() (see module wxFontData)`

Types:

```
This = wxFontDialog()
```

See **external documentation**.

`destroy(This::wxFontDialog()) -> ok`

Destroys this object, do not use object again

wxFontPickerCtrl

Erlang module

See external documentation: **wxFontPickerCtrl**.

This class is derived (and can use functions) from:
wxPickerBase
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxFontPickerCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFontPickerCtrl()`

See **external documentation**.

`new(Parent, Id) -> wxFontPickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxFontPickerCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {initial, wxFont() (see module wxFont)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
| {validator, wx_object() (see module wx)}
```

See **external documentation**.

`create(This, Parent, Id) -> boolean()`

Types:

```
This = wxFontPickerCtrl()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

create(This, Parent, Id, Option::[Option]) -> boolean()

Types:

This = wxFontPickerCtrl()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Option = {initial, wxFont() (see module wxFont)} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {validator, wx_object() (see module wx)}

See **external documentation**.

getSelectedFont(This) -> wxFont() (see module wxFont)

Types:

This = wxFontPickerCtrl()

See **external documentation**.

setSelectedFont(This, F) -> ok

Types:

This = wxFontPickerCtrl()

F = wxFont() (see module wxFont)

See **external documentation**.

getMaxPointSize(This) -> integer()

Types:

This = wxFontPickerCtrl()

See **external documentation**.

setMaxPointSize(This, Max) -> ok

Types:

This = wxFontPickerCtrl()

Max = integer()

See **external documentation**.

destroy(This::wxFontPickerCtrl()) -> ok

Destroys this object, do not use object again

wxFontPickerEvent

Erlang module

See external documentation: **wxFontPickerEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_fontpicker_changed

See also the message variant *#wxFontPicker{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxFontPickerEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getFont(This) -> wxFont() (see module wxFont)`

Types:

`This = wxFontPickerEvent()`

See **external documentation**.

wxFrame

Erlang module

See external documentation: **wxFrame**.

This class is derived (and can use functions) from:
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxFrame()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxFrame()`

See **external documentation**.

`new(Parent, Id, Title) -> wxFrame()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Title, [])*.

`new(Parent, Id, Title, Option::[Option]) -> wxFrame()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent, Id, Title) -> boolean()`

Types:

```
This = wxFrame()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Title, [])*.

create(This, Parent, Id, Title, Option::[Option]) -> boolean()

Types:

This = wxFrame()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Title = chardata() (see module unicode)

Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}

See **external documentation**.

createStatusBar(This) -> wxStatusBar() (see module wxStatusBar)

Types:

This = wxFrame()

Equivalent to *createStatusBar(This, [])*.

createStatusBar(This, Option::[Option]) -> wxStatusBar() (see module wxStatusBar)

Types:

This = wxFrame()

Option = {number, integer()} | {style, integer()} | {id, integer()}

See **external documentation**.

createToolBar(This) -> wxToolBar() (see module wxToolBar)

Types:

This = wxFrame()

Equivalent to *createToolBar(This, [])*.

createToolBar(This, Option::[Option]) -> wxToolBar() (see module wxToolBar)

Types:

This = wxFrame()

Option = {style, integer()} | {id, integer()}

See **external documentation**.

getClientAreaOrigin(This) -> {X::integer(), Y::integer()}

Types:

This = wxFrame()

See **external documentation**.

getMenuBar(This) -> wxMenuBar() (see module wxMenuBar)

Types:

This = wxFrame()

See **external documentation**.

getStatusBar(This) -> wxStatusBar() (see module wxStatusBar)

Types:

This = wxFrame()

See **external documentation**.

getStatusBarPane(This) -> integer()

Types:

This = wxFrame()

See **external documentation**.

getToolBar(This) -> wxToolBar() (see module wxToolBar)

Types:

This = wxFrame()

See **external documentation**.

processCommand(This, Winid) -> boolean()

Types:

This = wxFrame()

Winid = integer()

See **external documentation**.

sendSizeEvent(This) -> ok

Types:

This = wxFrame()

See **external documentation**.

setMenuBar(This, Menubar) -> ok

Types:

This = wxFrame()

Menubar = wxMenuBar() (see module wxMenuBar)

See **external documentation**.

setStatusBar(This, Statbar) -> ok

Types:

This = wxFrame()

Statbar = wxStatusBar() (see module wxStatusBar)

See **external documentation**.

setStatusBarPane(This, N) -> ok

Types:

This = wxFrame()

N = integer()

See **external documentation**.

setStatusText(This, Text) -> ok

Types:

This = wxFrame()
Text = chardata() (see module unicode)

Equivalent to *setStatusText(This, Text, [])*.

setStatusText(This, Text, Option::[Option]) -> ok

Types:

This = wxFrame()
Text = chardata() (see module unicode)
Option = {number, integer()}

See **external documentation**.

setStatusWidths(This, Widths_field) -> ok

Types:

This = wxFrame()
Widths_field = [integer()]

See **external documentation**.

setToolBar(This, Toolbar) -> ok

Types:

This = wxFrame()
Toolbar = wxToolBar() (see module wxToolBar)

See **external documentation**.

destroy(This::wxFrame()) -> ok

Destroys this object, do not use object again

wxGBSizerItem

Erlang module

See external documentation: **wxGBSizerItem**.

This class is derived (and can use functions) from:
wxSizerItem

DATA TYPES

`wxGBSizerItem()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxGLCanvas

Erlang module

See external documentation: **wxGLCanvas**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxGLCanvas()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent) -> wxGLCanvas()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Shared) -> wxGLCanvas()`

Types:

`Parent = wxWindow() (see module wxWindow)`
`Shared = wx_object() (see module wx) | wxGLCanvas()`

See **external documentation**.
Also:
new(Parent, [Option]) -> wxGLCanvas() when
Parent::wxWindow:wxWindow(),
Option :: {id, integer()}
| {pos, {X::integer(), Y::integer()}}
| {size, {W::integer(), H::integer()}}
| {style, integer()}
| {name, unicode:chardata()}
| {attribList, [integer()]}
| {palette, wxPalette:wxPalette()}.

`new(Parent, Shared, Option::[Option]) -> wxGLCanvas()`

Types:

`Parent = wxWindow() (see module wxWindow)`
`Shared = wx_object() (see module wx) | wxGLCanvas()`
`Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {name, chardata() (see module unicode)} | {attribList, [integer()]} | {palette, wxPalette() (see module wxPalette)}`

See **external documentation**.

`getContext(This) -> wx_object() (see module wx)`

Types:

`This = wxGLCanvas()`

See **external documentation**.

`setCurrent(This) -> ok`

Types:

`This = wxGLCanvas()`

See **external documentation**.

`swapBuffers(This) -> ok`

Types:

`This = wxGLCanvas()`

See **external documentation**.

`destroy(This::wxGLCanvas()) -> ok`

Destroys this object, do not use object again

wxGauge

Erlang module

See external documentation: **wxGauge**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxGauge()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGauge()`

See **external documentation**.

`new(Parent, Id, Range) -> wxGauge()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Range = integer()
```

Equivalent to *new(Parent, Id, Range, [])*.

`new(Parent, Id, Range, Option::[Option]) -> wxGauge()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Range = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

`create(This, Parent, Id, Range) -> boolean()`

Types:

```
This = wxGauge()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Range = integer()
```

Equivalent to *create(This, Parent, Id, Range, [])*.

create(This, Parent, Id, Range, Option::[Option]) -> boolean()

Types:

This = wxGauge()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Range = integer()

Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {validator, wx_object() (see module wx)}

See **external documentation**.

getBezelFace(This) -> integer()

Types:

This = wxGauge()

See **external documentation**.

getRange(This) -> integer()

Types:

This = wxGauge()

See **external documentation**.

getShadowWidth(This) -> integer()

Types:

This = wxGauge()

See **external documentation**.

getValue(This) -> integer()

Types:

This = wxGauge()

See **external documentation**.

isVertical(This) -> boolean()

Types:

This = wxGauge()

See **external documentation**.

setBezelFace(This, W) -> ok

Types:

This = wxGauge()

W = integer()

See **external documentation**.

setRange(This, R) -> ok

Types:

This = wxGauge()

R = integer()

See **external documentation**.

setShadowWidth(This, W) -> ok

Types:

This = wxGauge()

W = integer()

See **external documentation**.

setValue(This, Pos) -> ok

Types:

This = wxGauge()

Pos = integer()

See **external documentation**.

pulse(This) -> ok

Types:

This = wxGauge()

See **external documentation**.

destroy(This::wxGauge()) -> ok

Destroys this object, do not use object again

wxGenericDirCtrl

Erlang module

See external documentation: **wxGenericDirCtrl**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxGenericDirCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGenericDirCtrl()`

See **external documentation**.

`new(Parent) -> wxGenericDirCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxGenericDirCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {dir, chardata() (see module unicode)} | {pos,
{X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}}
| {style, integer()} | {filter, chardata() (see module unicode)} |
{defaultFilter, integer()}
```

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

```
This = wxGenericDirCtrl()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

```
This = wxGenericDirCtrl()
Parent = wxWindow() (see module wxWindow)
```

```
Option = {id, integer()} | {dir, chardata() (see module unicode)} | {pos,
{X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}}
| {style, integer()} | {filter, chardata() (see module unicode)} |
{defaultFilter, integer()}
```

See **external documentation**.

`init(This) -> ok`

Types:

```
This = wxGenericDirCtrl()
```

See **external documentation**.

`collapseTree(This) -> ok`

Types:

```
This = wxGenericDirCtrl()
```

See **external documentation**.

`expandPath(This, Path) -> boolean()`

Types:

```
This = wxGenericDirCtrl()
Path = chardata() (see module unicode)
```

See **external documentation**.

`getDefaultPath(This) -> charlist() (see module unicode)`

Types:

```
This = wxGenericDirCtrl()
```

See **external documentation**.

`getPath(This) -> charlist() (see module unicode)`

Types:

```
This = wxGenericDirCtrl()
```

See **external documentation**.

`getFilePath(This) -> charlist() (see module unicode)`

Types:

```
This = wxGenericDirCtrl()
```

See **external documentation**.

`getFilter(This) -> charlist() (see module unicode)`

Types:

```
This = wxGenericDirCtrl()
```

See **external documentation**.

getFilterIndex(This) -> integer()

Types:

This = wxGenericDirCtrl()

See **external documentation**.

getRootId(This) -> integer()

Types:

This = wxGenericDirCtrl()

See **external documentation**.

getTreeCtrl(This) -> wxTreeCtrl() (see module wxTreeCtrl)

Types:

This = wxGenericDirCtrl()

See **external documentation**.

reCreateTree(This) -> ok

Types:

This = wxGenericDirCtrl()

See **external documentation**.

setDefaultPath(This, Path) -> ok

Types:

This = wxGenericDirCtrl()
Path = chardata() (see module unicode)

See **external documentation**.

setFilter(This, Filter) -> ok

Types:

This = wxGenericDirCtrl()
Filter = chardata() (see module unicode)

See **external documentation**.

setFilterIndex(This, N) -> ok

Types:

This = wxGenericDirCtrl()
N = integer()

See **external documentation**.

setPath(This, Path) -> ok

Types:

This = wxGenericDirCtrl()
Path = chardata() (see module unicode)

See **external documentation**.

destroy(This::wxGenericDirCtrl()) -> ok

Destroys this object, do not use object again

wxGraphicsBrush

Erlang module

See external documentation: **wxGraphicsBrush**.

This class is derived (and can use functions) from:
wxGraphicsObject

DATA TYPES

`wxGraphicsBrush()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxGraphicsContext

Erlang module

See external documentation: **wxGraphicsContext**.

This class is derived (and can use functions) from:
wxGraphicsObject

DATA TYPES

`wxGraphicsContext()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`create() -> wxGraphicsContext()`

See **external documentation**.

`create(Dc) -> wxGraphicsContext()`

Types:

`Dc = wxWindowDC() (see module wxWindowDC) | wxWindow() (see module wxWindow)`

See **external documentation**.

`createPen(This, Pen) -> wxGraphicsPen() (see module wxGraphicsPen)`

Types:

`This = wxGraphicsContext()`

`Pen = wxPen() (see module wxPen)`

See **external documentation**.

`createBrush(This, Brush) -> wxGraphicsBrush() (see module wxGraphicsBrush)`

Types:

`This = wxGraphicsContext()`

`Brush = wxBrush() (see module wxBrush)`

See **external documentation**.

`createRadialGradientBrush(This, Xo, Yo, Xc, Yc, Radius, OColor, CColor) -> wxGraphicsBrush() (see module wxGraphicsBrush)`

Types:

`This = wxGraphicsContext()`

`Xo = number()`

`Yo = number()`

`Xc = number()`

```
Yc = number()
Radius = number()
OColor = wx_colour() (see module wx)
CColor = wx_colour() (see module wx)
```

See **external documentation**.

createLinearGradientBrush(This, X1, Y1, X2, Y2, C1, C2) -> wxGraphicsBrush() (see module wxGraphicsBrush)

Types:

```
This = wxGraphicsContext()
X1 = number()
Y1 = number()
X2 = number()
Y2 = number()
C1 = wx_colour() (see module wx)
C2 = wx_colour() (see module wx)
```

See **external documentation**.

createFont(This, Font) -> wxGraphicsFont() (see module wxGraphicsFont)

Types:

```
This = wxGraphicsContext()
Font = wxFont() (see module wxFont)
```

Equivalent to *createFont(This, Font, [])*.

createFont(This, Font, Option::[Option]) -> wxGraphicsFont() (see module wxGraphicsFont)

Types:

```
This = wxGraphicsContext()
Font = wxFont() (see module wxFont)
Option = {col, wx_colour() (see module wx)}
```

See **external documentation**.

createMatrix(This) -> wxGraphicsMatrix() (see module wxGraphicsMatrix)

Types:

```
This = wxGraphicsContext()
```

Equivalent to *createMatrix(This, [])*.

createMatrix(This, Option::[Option]) -> wxGraphicsMatrix() (see module wxGraphicsMatrix)

Types:

```
This = wxGraphicsContext()
Option = {a, number()} | {b, number()} | {c, number()} | {d, number()} |
{tx, number()} | {ty, number()}
```

See **external documentation**.

createPath(This) -> wxGraphicsPath() (see module wxGraphicsPath)

Types:

This = wxGraphicsContext()

See **external documentation**.

clip(This, Region) -> ok

Types:

This = wxGraphicsContext()
Region = wxRegion() (see module wxRegion)

See **external documentation**.

clip(This, X, Y, W, H) -> ok

Types:

This = wxGraphicsContext()
X = number()
Y = number()
W = number()
H = number()

See **external documentation**.

resetClip(This) -> ok

Types:

This = wxGraphicsContext()

See **external documentation**.

drawBitmap(This, Bmp, X, Y, W, H) -> ok

Types:

This = wxGraphicsContext()
Bmp = wxBitmap() (see module wxBitmap)
X = number()
Y = number()
W = number()
H = number()

See **external documentation**.

drawEllipse(This, X, Y, W, H) -> ok

Types:

This = wxGraphicsContext()
X = number()
Y = number()
W = number()

```
H = number()
```

See **external documentation**.

```
drawIcon(This, Icon, X, Y, W, H) -> ok
```

Types:

```
This = wxGraphicsContext()
Icon = wxIcon() (see module wxIcon)
X = number()
Y = number()
W = number()
H = number()
```

See **external documentation**.

```
drawLines(This, Points) -> ok
```

Types:

```
This = wxGraphicsContext()
Points = [{X::float(), Y::float()}]
```

Equivalent to *drawLines(This, Points, [])*.

```
drawLines(This, Points, Option::[Option]) -> ok
```

Types:

```
This = wxGraphicsContext()
Points = [{X::float(), Y::float()}]
Option = {fillStyle, integer()}
```

See **external documentation**.

```
drawPath(This, Path) -> ok
```

Types:

```
This = wxGraphicsContext()
Path = wxGraphicsPath() (see module wxGraphicsPath)
```

Equivalent to *drawPath(This, Path, [])*.

```
drawPath(This, Path, Option::[Option]) -> ok
```

Types:

```
This = wxGraphicsContext()
Path = wxGraphicsPath() (see module wxGraphicsPath)
Option = {fillStyle, integer()}
```

See **external documentation**.

```
drawRectangle(This, X, Y, W, H) -> ok
```

Types:

```
This = wxGraphicsContext()
X = number()
```

```
Y = number()
W = number()
H = number()
```

See **external documentation**.

drawRoundedRectangle(This, X, Y, W, H, Radius) -> ok

Types:

```
This = wxGraphicsContext()
X = number()
Y = number()
W = number()
H = number()
Radius = number()
```

See **external documentation**.

drawText(This, Str, X, Y) -> ok

Types:

```
This = wxGraphicsContext()
Str = chardata() (see module unicode)
X = number()
Y = number()
```

See **external documentation**.

drawText(This, Str, X, Y, Angle) -> ok

Types:

```
This = wxGraphicsContext()
Str = chardata() (see module unicode)
X = number()
Y = number()
Angle = number()
```

See **external documentation**.
Also:
drawText(This, Str, X, Y, BackgroundBrush) -> ok when This::wxGraphicsContext(), Str::unicode:chardata(), X::number(), Y::number(), BackgroundBrush::wxGraphicsBrush:wxGraphicsBrush().

drawText(This, Str, X, Y, Angle, BackgroundBrush) -> ok

Types:

```
This = wxGraphicsContext()
Str = chardata() (see module unicode)
X = number()
Y = number()
Angle = number()
BackgroundBrush = wxGraphicsBrush() (see module wxGraphicsBrush)
```

See **external documentation**.

fillPath(This, Path) -> ok

Types:

```
This = wxGraphicsContext()
Path = wxGraphicsPath() (see module wxGraphicsPath)
```

Equivalent to *fillPath(This, Path, [])*.

fillPath(This, Path, Option::[Option]) -> ok

Types:

```
This = wxGraphicsContext()
Path = wxGraphicsPath() (see module wxGraphicsPath)
Option = {fillStyle, integer()}
```

See **external documentation**.

strokePath(This, Path) -> ok

Types:

```
This = wxGraphicsContext()
Path = wxGraphicsPath() (see module wxGraphicsPath)
```

See **external documentation**.

getPartialTextExtents(This, Text) -> [number()]

Types:

```
This = wxGraphicsContext()
Text = chardata() (see module unicode)
```

See **external documentation**.

getTextExtent(This, Text) -> Result

Types:

```
Result = {Width::number(), Height::number(), Descent::number(),
ExternalLeading::number()}
This = wxGraphicsContext()
Text = chardata() (see module unicode)
```

See **external documentation**.

rotate(This, Angle) -> ok

Types:

```
This = wxGraphicsContext()
Angle = number()
```

See **external documentation**.

scale(This, XScale, YScale) -> ok

Types:

```
This = wxGraphicsContext()
XScale = number()
YScale = number()
```

See **external documentation**.

translate(This, Dx, Dy) -> ok

Types:

```
This = wxGraphicsContext()
Dx = number()
Dy = number()
```

See **external documentation**.

getTransform(This) -> wxGraphicsMatrix() (see module wxGraphicsMatrix)

Types:

```
This = wxGraphicsContext()
```

See **external documentation**.

setTransform(This, Matrix) -> ok

Types:

```
This = wxGraphicsContext()
Matrix = wxGraphicsMatrix() (see module wxGraphicsMatrix)
```

See **external documentation**.

concatTransform(This, Matrix) -> ok

Types:

```
This = wxGraphicsContext()
Matrix = wxGraphicsMatrix() (see module wxGraphicsMatrix)
```

See **external documentation**.

setBrush(This, Brush) -> ok

Types:

```
This = wxGraphicsContext()
Brush = wxGraphicsBrush() (see module wxGraphicsBrush) | wxBrush() (see
module wxBrush)
```

See **external documentation**.

setFont(This, Font) -> ok

Types:

```
This = wxGraphicsContext()
Font = wxGraphicsFont() (see module wxGraphicsFont)
```

See **external documentation**.

setFont(This, Font, Colour) -> ok

Types:

This = wxGraphicsContext()
Font = wxFont() (see module wxFont)
Colour = wx_colour() (see module wx)

See **external documentation**.

setPen(This, Pen) -> ok

Types:

This = wxGraphicsContext()
Pen = wxPen() (see module wxPen) | wxGraphicsPen() (see module wxGraphicsPen)

See **external documentation**.

strokeLine(This, X1, Y1, X2, Y2) -> ok

Types:

This = wxGraphicsContext()
X1 = number()
Y1 = number()
X2 = number()
Y2 = number()

See **external documentation**.

strokeLines(This, Points) -> ok

Types:

This = wxGraphicsContext()
Points = [{X::float(), Y::float()}]

See **external documentation**.

destroy(This::wxGraphicsContext()) -> ok

Destroys this object, do not use object again

wxGraphicsFont

Erlang module

See external documentation: **wxGraphicsFont**.

This class is derived (and can use functions) from:
wxGraphicsObject

DATA TYPES

`wxGraphicsFont()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxGraphicsMatrix

Erlang module

See external documentation: **wxGraphicsMatrix**.

This class is derived (and can use functions) from:
wxGraphicsObject

DATA TYPES

`wxGraphicsMatrix()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`concat(This, T) -> ok`

Types:

```
This = wxGraphicsMatrix()
T = wxGraphicsMatrix()
```

See **external documentation**.

`get(This) -> Result`

Types:

```
Result = {A::number(), B::number(), C::number(), D::number(),
Tx::number(), Ty::number()}
This = wxGraphicsMatrix()
```

See **external documentation**.

`invert(This) -> ok`

Types:

```
This = wxGraphicsMatrix()
```

See **external documentation**.

`isEqual(This, T) -> boolean()`

Types:

```
This = wxGraphicsMatrix()
T = wxGraphicsMatrix()
```

See **external documentation**.

`isIdentity(This) -> boolean()`

Types:

```
This = wxGraphicsMatrix()
```

See **external documentation**.

rotate(This, Angle) -> ok

Types:

This = wxGraphicsMatrix()
Angle = number()

See **external documentation**.

scale(This, XScale, YScale) -> ok

Types:

This = wxGraphicsMatrix()
XScale = number()
YScale = number()

See **external documentation**.

translate(This, Dx, Dy) -> ok

Types:

This = wxGraphicsMatrix()
Dx = number()
Dy = number()

See **external documentation**.

set(This) -> ok

Types:

This = wxGraphicsMatrix()

Equivalent to *set(This, [])*.

set(This, Option::[Option]) -> ok

Types:

This = wxGraphicsMatrix()
Option = {a, number()} | {b, number()} | {c, number()} | {d, number()} | {tx, number()} | {ty, number()}

See **external documentation**.

transformPoint(This) -> {X::number(), Y::number()}

Types:

This = wxGraphicsMatrix()

See **external documentation**.

transformDistance(This) -> {Dx::number(), Dy::number()}

Types:

This = wxGraphicsMatrix()

See **external documentation**.

wxGraphicsObject

Erlang module

See external documentation: **wxGraphicsObject**.

DATA TYPES

`wxGraphicsObject()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getRenderer(This) -> wxGraphicsRenderer() (see module wxGraphicsRenderer)`

Types:

`This = wxGraphicsObject()`

See **external documentation**.

`isNull(This) -> boolean()`

Types:

`This = wxGraphicsObject()`

See **external documentation**.

`destroy(This::wxGraphicsObject()) -> ok`

Destroys this object, do not use object again

wxGraphicsPath

Erlang module

See external documentation: **wxGraphicsPath**.

This class is derived (and can use functions) from:
wxGraphicsObject

DATA TYPES

`wxGraphicsPath()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`moveToPoint(This, P) -> ok`

Types:

```
This = wxGraphicsPath()
P = {X::float(), Y::float()}
```

See **external documentation**.

`moveToPoint(This, X, Y) -> ok`

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
```

See **external documentation**.

`addArc(This, C, R, StartAngle, EndAngle, Clockwise) -> ok`

Types:

```
This = wxGraphicsPath()
C = {X::float(), Y::float()}
R = number()
StartAngle = number()
EndAngle = number()
Clockwise = boolean()
```

See **external documentation**.

`addArc(This, X, Y, R, StartAngle, EndAngle, Clockwise) -> ok`

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
```

```
R = number()
StartAngle = number()
EndAngle = number()
Clockwise = boolean()
```

See **external documentation**.

addArcToPoint(This, X1, Y1, X2, Y2, R) -> ok

Types:

```
This = wxGraphicsPath()
X1 = number()
Y1 = number()
X2 = number()
Y2 = number()
R = number()
```

See **external documentation**.

addCircle(This, X, Y, R) -> ok

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
R = number()
```

See **external documentation**.

addCurveToPoint(This, C1, C2, E) -> ok

Types:

```
This = wxGraphicsPath()
C1 = {X::float(), Y::float()}
C2 = {X::float(), Y::float()}
E = {X::float(), Y::float()}
```

See **external documentation**.

addCurveToPoint(This, Cx1, Cy1, Cx2, Cy2, X, Y) -> ok

Types:

```
This = wxGraphicsPath()
Cx1 = number()
Cy1 = number()
Cx2 = number()
Cy2 = number()
X = number()
Y = number()
```

See **external documentation**.

addEllipse(This, X, Y, W, H) -> ok

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
W = number()
H = number()
```

See **external documentation**.

addLineToPoint(This, P) -> ok

Types:

```
This = wxGraphicsPath()
P = {X::float(), Y::float()}
```

See **external documentation**.

addLineToPoint(This, X, Y) -> ok

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
```

See **external documentation**.

addPath(This, Path) -> ok

Types:

```
This = wxGraphicsPath()
Path = wxGraphicsPath()
```

See **external documentation**.

addQuadCurveToPoint(This, Cx, Cy, X, Y) -> ok

Types:

```
This = wxGraphicsPath()
Cx = number()
Cy = number()
X = number()
Y = number()
```

See **external documentation**.

addRectangle(This, X, Y, W, H) -> ok

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
W = number()
```

```
H = number()
```

See **external documentation**.

addRoundedRectangle(This, X, Y, W, H, Radius) -> ok

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
W = number()
H = number()
Radius = number()
```

See **external documentation**.

closeSubpath(This) -> ok

Types:

```
This = wxGraphicsPath()
```

See **external documentation**.

contains(This, C) -> boolean()

Types:

```
This = wxGraphicsPath()
C = {X::float(), Y::float()}
```

Equivalent to *contains(This, C, [])*.

contains(This, X, Y) -> boolean()

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
```

See **external documentation**.
Also:
contains(This, C, [Option]) -> boolean() when
This::wxGraphicsPath(), C::{X::float(), Y::float()},
Option :: {fillStyle, integer()}.

contains(This, X, Y, Option::[Option]) -> boolean()

Types:

```
This = wxGraphicsPath()
X = number()
Y = number()
Option = {fillStyle, integer()}
```

See **external documentation**.

getBox(This) -> {X::float(), Y::float(), W::float(), H::float()}

Types:

This = wxGraphicsPath()

See **external documentation**.

getCurrentPoint(This) -> {X::float(), Y::float()}

Types:

This = wxGraphicsPath()

See **external documentation**.

transform(This, Matrix) -> ok

Types:

This = wxGraphicsPath()

Matrix = wxGraphicsMatrix() (see module wxGraphicsMatrix)

See **external documentation**.

wxGraphicsPen

Erlang module

See external documentation: **wxGraphicsPen**.

This class is derived (and can use functions) from:
wxGraphicsObject

DATA TYPES

`wxGraphicsPen()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxGraphicsRenderer

Erlang module

See external documentation: **wxGraphicsRenderer**.

DATA TYPES

`wxGraphicsRenderer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getDefaultRenderer() -> wxGraphicsRenderer()`

See **external documentation**.

`createContext(This, Dc) -> wxGraphicsContext() (see module wxGraphicsContext)`

Types:

`This = wxGraphicsRenderer()`

`Dc = wxWindowDC() (see module wxWindowDC) | wxWindow() (see module wxWindow)`

See **external documentation**.

`createPen(This, Pen) -> wxGraphicsPen() (see module wxGraphicsPen)`

Types:

`This = wxGraphicsRenderer()`

`Pen = wxPen() (see module wxPen)`

See **external documentation**.

`createBrush(This, Brush) -> wxGraphicsBrush() (see module wxGraphicsBrush)`

Types:

`This = wxGraphicsRenderer()`

`Brush = wxBrush() (see module wxBrush)`

See **external documentation**.

`createLinearGradientBrush(This, X1, Y1, X2, Y2, C1, C2) -> wxGraphicsBrush() (see module wxGraphicsBrush)`

Types:

`This = wxGraphicsRenderer()`

`X1 = number()`

`Y1 = number()`

`X2 = number()`

`Y2 = number()`

```
C1 = wx_colour() (see module wx)
C2 = wx_colour() (see module wx)
```

See **external documentation**.

createRadialGradientBrush(This, Xo, Yo, Xc, Yc, Radius, OColor, CColor) -> wxGraphicsBrush() (see module wxGraphicsBrush)

Types:

```
This = wxGraphicsRenderer()
Xo = number()
Yo = number()
Xc = number()
Yc = number()
Radius = number()
OColor = wx_colour() (see module wx)
CColor = wx_colour() (see module wx)
```

See **external documentation**.

createFont(This, Font) -> wxGraphicsFont() (see module wxGraphicsFont)

Types:

```
This = wxGraphicsRenderer()
Font = wxFont() (see module wxFont)
```

Equivalent to *createFont(This, Font, [])*.

createFont(This, Font, Option::[Option]) -> wxGraphicsFont() (see module wxGraphicsFont)

Types:

```
This = wxGraphicsRenderer()
Font = wxFont() (see module wxFont)
Option = {col, wx_colour() (see module wx)}
```

See **external documentation**.

createMatrix(This) -> wxGraphicsMatrix() (see module wxGraphicsMatrix)

Types:

```
This = wxGraphicsRenderer()
```

Equivalent to *createMatrix(This, [])*.

createMatrix(This, Option::[Option]) -> wxGraphicsMatrix() (see module wxGraphicsMatrix)

Types:

```
This = wxGraphicsRenderer()
Option = {a, number()} | {b, number()} | {c, number()} | {d, number()} |
{tx, number()} | {ty, number()}
```

See **external documentation**.

createPath(This) -> wxGraphicsPath() (see module wxGraphicsPath)

Types:

This = wxGraphicsRenderer()

See **external documentation**.

wxGrid

Erlang module

See external documentation: **wxGrid**.

This class is derived (and can use functions) from:
wxScrolledWindow
wxPanel
wxWindow
wxEvtHandler

DATA TYPES

`wxGrid()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGrid()`

See **external documentation**.

`new(Parent, Id) -> wxGrid()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, X, Y) -> wxGrid()`

Types:

```
Parent = wxWindow() (see module wxWindow)
X = integer()
Y = integer()
```

See **external documentation**.
Also:
new(Parent, Id, [Option]) -> wxGrid() when
Parent::wxWindow:wxWindow(), Id::integer(),
Option :: {pos, {X::integer(), Y::integer()}}
| {size, {W::integer(), H::integer()}}
| {style, integer()}.

`new(Parent, X, Y, Option::[Option]) -> wxGrid()`

Types:

```
Parent = wxWindow() (see module wxWindow)
X = integer()
Y = integer()
```

```
Option = {w, integer()} | {h, integer()} | {style, integer()}
```

See **external documentation**.

appendCols(This) -> boolean()

Types:

```
This = wxGrid()
```

Equivalent to *appendCols(This, [])*.

appendCols(This, Option::[Option]) -> boolean()

Types:

```
This = wxGrid()
Option = {numCols, integer()} | {updateLabels, boolean()}
```

See **external documentation**.

appendRows(This) -> boolean()

Types:

```
This = wxGrid()
```

Equivalent to *appendRows(This, [])*.

appendRows(This, Option::[Option]) -> boolean()

Types:

```
This = wxGrid()
Option = {numRows, integer()} | {updateLabels, boolean()}
```

See **external documentation**.

autoSize(This) -> ok

Types:

```
This = wxGrid()
```

See **external documentation**.

autoSizeColumn(This, Col) -> ok

Types:

```
This = wxGrid()
Col = integer()
```

Equivalent to *autoSizeColumn(This, Col, [])*.

autoSizeColumn(This, Col, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Col = integer()
Option = {setAsMin, boolean()}
```

See **external documentation**.

autoSizeColumns(This) -> ok

Types:

This = wxGrid()

Equivalent to *autoSizeColumns(This, [])*.

autoSizeColumns(This, Option::[Option]) -> ok

Types:

This = wxGrid()
Option = {setAsMin, boolean()}

See **external documentation**.

autoSizeRow(This, Row) -> ok

Types:

This = wxGrid()
Row = integer()

Equivalent to *autoSizeRow(This, Row, [])*.

autoSizeRow(This, Row, Option::[Option]) -> ok

Types:

This = wxGrid()
Row = integer()
Option = {setAsMin, boolean()}

See **external documentation**.

autoSizeRows(This) -> ok

Types:

This = wxGrid()

Equivalent to *autoSizeRows(This, [])*.

autoSizeRows(This, Option::[Option]) -> ok

Types:

This = wxGrid()
Option = {setAsMin, boolean()}

See **external documentation**.

beginBatch(This) -> ok

Types:

This = wxGrid()

See **external documentation**.

blockToDeviceRect(This, TopLeft, BottomRight) -> {X::integer(), Y::integer(), W::integer(), H::integer()}

Types:

```
This = wxGrid()
TopLeft = {R::integer(), C::integer()}
BottomRight = {R::integer(), C::integer()}
```

See **external documentation**.

canDragColSize(This) -> boolean()

Types:

```
This = wxGrid()
```

See **external documentation**.

canDragRowSize(This) -> boolean()

Types:

```
This = wxGrid()
```

See **external documentation**.

canDragGridSize(This) -> boolean()

Types:

```
This = wxGrid()
```

See **external documentation**.

canEnableCellControl(This) -> boolean()

Types:

```
This = wxGrid()
```

See **external documentation**.

cellToRect(This, Coords) -> {X::integer(), Y::integer(), W::integer(), H::integer()}

Types:

```
This = wxGrid()
Coords = {R::integer(), C::integer()}
```

See **external documentation**.

cellToRect(This, Row, Col) -> {X::integer(), Y::integer(), W::integer(), H::integer()}

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
```

See **external documentation**.

clearGrid(This) -> ok

Types:

```
This = wxGrid()
```

See **external documentation**.

clearSelection(This) -> ok

Types:

This = wxGrid()

See **external documentation**.

createGrid(This, NumRows, NumCols) -> boolean()

Types:

This = wxGrid()
NumRows = integer()
NumCols = integer()

Equivalent to *createGrid(This, NumRows, NumCols, [])*.

createGrid(This, NumRows, NumCols, Option::[Option]) -> boolean()

Types:

This = wxGrid()
NumRows = integer()
NumCols = integer()
Option = {selmode, wx_enum() (see module wx)}

See **external documentation**.
Selmode = ?wxGrid_wxGridSelectCells | ?wxGrid_wxGridSelectRows | ?wxGrid_wxGridSelectColumns

deleteCols(This) -> boolean()

Types:

This = wxGrid()

Equivalent to *deleteCols(This, [])*.

deleteCols(This, Option::[Option]) -> boolean()

Types:

This = wxGrid()
Option = {pos, integer()} | {numCols, integer()} | {updateLabels, boolean()}

See **external documentation**.

deleteRows(This) -> boolean()

Types:

This = wxGrid()

Equivalent to *deleteRows(This, [])*.

deleteRows(This, Option::[Option]) -> boolean()

Types:

This = wxGrid()

```
Option = {pos, integer()} | {numRows, integer()} | {updateLabels,
boolean()}
```

See **external documentation**.

disableCellEditControl(This) -> ok

Types:

```
This = wxGrid()
```

See **external documentation**.

disableDragColSize(This) -> ok

Types:

```
This = wxGrid()
```

See **external documentation**.

disableDragGridSize(This) -> ok

Types:

```
This = wxGrid()
```

See **external documentation**.

disableDragRowSize(This) -> ok

Types:

```
This = wxGrid()
```

See **external documentation**.

enableCellEditControl(This) -> ok

Types:

```
This = wxGrid()
```

Equivalent to *enableCellEditControl(This, [])*.

enableCellEditControl(This, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Option = {enable, boolean()}
```

See **external documentation**.

enableDragColSize(This) -> ok

Types:

```
This = wxGrid()
```

Equivalent to *enableDragColSize(This, [])*.

enableDragColSize(This, Option::[Option]) -> ok

Types:

```
This = wxGrid()
```

```
Option = {enable, boolean()}
```

See **external documentation**.

enableDragGridSize(This) -> ok

Types:

```
This = wxGrid()
```

Equivalent to *enableDragGridSize(This, [])*.

enableDragGridSize(This, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Option = {enable, boolean()}
```

See **external documentation**.

enableDragRowSize(This) -> ok

Types:

```
This = wxGrid()
```

Equivalent to *enableDragRowSize(This, [])*.

enableDragRowSize(This, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Option = {enable, boolean()}
```

See **external documentation**.

enableEditing(This, Edit) -> ok

Types:

```
This = wxGrid()
Edit = boolean()
```

See **external documentation**.

enableGridLines(This) -> ok

Types:

```
This = wxGrid()
```

Equivalent to *enableGridLines(This, [])*.

enableGridLines(This, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Option = {enable, boolean()}
```

See **external documentation**.

endBatch(This) -> ok

Types:

This = wxGrid()

See **external documentation**.

fit(This) -> ok

Types:

This = wxGrid()

See **external documentation**.

forceRefresh(This) -> ok

Types:

This = wxGrid()

See **external documentation**.

getBatchCount(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getCellAlignment(This, Row, Col) -> {Horiz::integer(), Vert::integer()}

Types:

This = wxGrid()
Row = integer()
Col = integer()

See **external documentation**.

getCellBackgroundColour(This, Row, Col) -> wx_colour4() (see module wx)

Types:

This = wxGrid()
Row = integer()
Col = integer()

See **external documentation**.

getCellEditor(This, Row, Col) -> wxGridCellEditor() (see module wxGridCellEditor)

Types:

This = wxGrid()
Row = integer()
Col = integer()

See **external documentation**.

getCellFont(This, Row, Col) -> wxFont() (see module wxFont)

Types:

This = wxGrid()
Row = integer()
Col = integer()

See **external documentation**.

getCellRenderer(This, Row, Col) -> wxGridCellRenderer() (see module wxGridCellRenderer)

Types:

This = wxGrid()
Row = integer()
Col = integer()

See **external documentation**.

getCellTextColour(This, Row, Col) -> wx_colour4() (see module wx)

Types:

This = wxGrid()
Row = integer()
Col = integer()

See **external documentation**.

getCellValue(This, Coords) -> charlist() (see module unicode)

Types:

This = wxGrid()
Coords = {R::integer(), C::integer()}

See **external documentation**.

getCellValue(This, Row, Col) -> charlist() (see module unicode)

Types:

This = wxGrid()
Row = integer()
Col = integer()

See **external documentation**.

getColLabelAlignment(This) -> {Horiz::integer(), Vert::integer()}

Types:

This = wxGrid()

See **external documentation**.

getColLabelSize(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getColLabelValue(This, Col) -> charlist() (see module unicode)

Types:

This = wxGrid()

Col = integer()

See **external documentation**.

getColMinimalAcceptableWidth(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getDefaultCellAlignment(This) -> {Horiz::integer(), Vert::integer()}

Types:

This = wxGrid()

See **external documentation**.

getDefaultCellBackgroundColour(This) -> wx_colour4() (see module wx)

Types:

This = wxGrid()

See **external documentation**.

getDefaultCellFont(This) -> wxFont() (see module wxFont)

Types:

This = wxGrid()

See **external documentation**.

getDefaultCellTextColour(This) -> wx_colour4() (see module wx)

Types:

This = wxGrid()

See **external documentation**.

getDefaultColLabelSize(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getDefaultColSize(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getDefaultEditor(This) -> wxGridCellEditor() (see module wxGridCellEditor)

Types:

This = wxGrid()

See **external documentation**.

getDefaultEditorForCell(This, C) -> wxGridCellEditor() (see module wxGridCellEditor)

Types:

This = wxGrid()

C = {R::integer(), C::integer()}

See **external documentation**.

getDefaultEditorForCell(This, Row, Col) -> wxGridCellEditor() (see module wxGridCellEditor)

Types:

This = wxGrid()

Row = integer()

Col = integer()

See **external documentation**.

getDefaultEditorForType(This, TypeName) -> wxGridCellEditor() (see module wxGridCellEditor)

Types:

This = wxGrid()

TypeName = chardata() (see module unicode)

See **external documentation**.

getDefaultRenderer(This) -> wxGridCellRenderer() (see module wxGridCellRenderer)

Types:

This = wxGrid()

See **external documentation**.

getDefaultRendererForCell(This, Row, Col) -> wxGridCellRenderer() (see module wxGridCellRenderer)

Types:

This = wxGrid()

Row = integer()

Col = integer()

See **external documentation**.

getDefaultRendererForType(This, TypeName) -> wxGridCellRenderer() (see module wxGridCellRenderer)

Types:

```
This = wxGrid()
TypeName = chardata() (see module unicode)
```

See **external documentation**.

getDefaultRowLabelSize(This) -> integer()

Types:

```
This = wxGrid()
```

See **external documentation**.

getDefaultRowSize(This) -> integer()

Types:

```
This = wxGrid()
```

See **external documentation**.

getGridCursorCol(This) -> integer()

Types:

```
This = wxGrid()
```

See **external documentation**.

getGridCursorRow(This) -> integer()

Types:

```
This = wxGrid()
```

See **external documentation**.

getGridLineColour(This) -> wx_colour4() (see module wx)

Types:

```
This = wxGrid()
```

See **external documentation**.

gridLinesEnabled(This) -> boolean()

Types:

```
This = wxGrid()
```

See **external documentation**.

getLabelBackgroundColour(This) -> wx_colour4() (see module wx)

Types:

```
This = wxGrid()
```

See **external documentation**.

getLabelFont(This) -> wxFont() (see module wxFont)

Types:

```
This = wxGrid()
```

See **external documentation**.

getLabelTextColour(This) -> wx_colour4() (see module wx)

Types:

This = wxGrid()

See **external documentation**.

getNumberCols(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getNumberRows(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getOrCreateCellAttr(This, Row, Col) -> wxGridCellAttr() (see module wxGridCellAttr)

Types:

This = wxGrid()

Row = integer()

Col = integer()

See **external documentation**.

getRowMinimalAcceptableHeight(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getRowLabelAlignment(This) -> {Horiz::integer(), Vert::integer()}

Types:

This = wxGrid()

See **external documentation**.

getRowLabelSize(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getRowLabelValue(This, Row) -> charlist() (see module unicode)

Types:

This = wxGrid()

Row = integer()

See **external documentation**.

getRowSize(This, Row) -> integer()

Types:

This = wxGrid()

Row = integer()

See **external documentation**.

getScrollLineX(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getScrollLineY(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getSelectedCells(This) -> [{R::integer(), C::integer()}]

Types:

This = wxGrid()

See **external documentation**.

getSelectedCols(This) -> [integer()]

Types:

This = wxGrid()

See **external documentation**.

getSelectedRows(This) -> [integer()]

Types:

This = wxGrid()

See **external documentation**.

getSelectionBackground(This) -> wx_colour4() (see module wx)

Types:

This = wxGrid()

See **external documentation**.

getSelectionBlockTopLeft(This) -> [{R::integer(), C::integer()}]

Types:

This = wxGrid()

See **external documentation**.

getSelectionBlockBottomRight(This) -> [{R::integer(), C::integer()}]

Types:

This = wxGrid()

See **external documentation**.

getSelectionForeground(This) -> wx_colour4() (see module wx)

Types:

This = wxGrid()

See **external documentation**.

getViewWidth(This) -> integer()

Types:

This = wxGrid()

See **external documentation**.

getGridWindow(This) -> wxWindow() (see module wxWindow)

Types:

This = wxGrid()

See **external documentation**.

getGridRowLabelWindow(This) -> wxWindow() (see module wxWindow)

Types:

This = wxGrid()

See **external documentation**.

getGridColLabelWindow(This) -> wxWindow() (see module wxWindow)

Types:

This = wxGrid()

See **external documentation**.

getGridCornerLabelWindow(This) -> wxWindow() (see module wxWindow)

Types:

This = wxGrid()

See **external documentation**.

hideCellEditControl(This) -> ok

Types:

This = wxGrid()

See **external documentation**.

insertCols(This) -> boolean()

Types:

This = wxGrid()

Equivalent to *insertCols(This, [])*.

insertCols(This, Option::[Option]) -> boolean()

Types:

This = wxGrid()

Option = {pos, integer()} | {numCols, integer()} | {updateLabels, boolean()}

See **external documentation**.

insertRows(This) -> boolean()

Types:

This = wxGrid()

Equivalent to *insertRows(This, [])*.

insertRows(This, Option::[Option]) -> boolean()

Types:

This = wxGrid()

Option = {pos, integer()} | {numRows, integer()} | {updateLabels, boolean()}

See **external documentation**.

isCellEditControlEnabled(This) -> boolean()

Types:

This = wxGrid()

See **external documentation**.

isCurrentCellReadOnly(This) -> boolean()

Types:

This = wxGrid()

See **external documentation**.

isEditable(This) -> boolean()

Types:

This = wxGrid()

See **external documentation**.

isInSelection(This, Coords) -> boolean()

Types:

This = wxGrid()

Coords = {R::integer(), C::integer()}

See **external documentation**.

isInSelection(This, Row, Col) -> boolean()

Types:

This = wxGrid()

```
Row = integer()
Col = integer()
```

See **external documentation**.

isReadOnly(This, Row, Col) -> boolean()

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
```

See **external documentation**.

isSelection(This) -> boolean()

Types:

```
This = wxGrid()
```

See **external documentation**.

isVisible(This, Coords) -> boolean()

Types:

```
This = wxGrid()
Coords = {R::integer(), C::integer()}
```

Equivalent to *isVisible(This, Coords, [])*.

isVisible(This, Row, Col) -> boolean()

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
```

See **external documentation**.
Also:
isVisible(This, Coords, [Option]) -> boolean() when
This::wxGrid(), Coords::{R::integer(), C::integer()},
Option :: {wholeCellVisible, boolean()}.

isVisible(This, Row, Col, Option::[Option]) -> boolean()

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
Option = {wholeCellVisible, boolean()}
```

See **external documentation**.

makeCellVisible(This, Coords) -> ok

Types:

```
This = wxGrid()
```

```
Coords = {R::integer(), C::integer()}
```

See **external documentation**.

```
makeCellVisible(This, Row, Col) -> ok
```

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
```

See **external documentation**.

```
moveCursorDown(This, ExpandSelection) -> boolean()
```

Types:

```
This = wxGrid()
ExpandSelection = boolean()
```

See **external documentation**.

```
moveCursorLeft(This, ExpandSelection) -> boolean()
```

Types:

```
This = wxGrid()
ExpandSelection = boolean()
```

See **external documentation**.

```
moveCursorRight(This, ExpandSelection) -> boolean()
```

Types:

```
This = wxGrid()
ExpandSelection = boolean()
```

See **external documentation**.

```
moveCursorUp(This, ExpandSelection) -> boolean()
```

Types:

```
This = wxGrid()
ExpandSelection = boolean()
```

See **external documentation**.

```
moveCursorDownBlock(This, ExpandSelection) -> boolean()
```

Types:

```
This = wxGrid()
ExpandSelection = boolean()
```

See **external documentation**.

```
moveCursorLeftBlock(This, ExpandSelection) -> boolean()
```

Types:

```
This = wxGrid()
```

```
ExpandSelection = boolean()
```

See **external documentation**.

moveCursorRightBlock(This, ExpandSelection) -> boolean()

Types:

```
This = wxGrid()
ExpandSelection = boolean()
```

See **external documentation**.

moveCursorUpBlock(This, ExpandSelection) -> boolean()

Types:

```
This = wxGrid()
ExpandSelection = boolean()
```

See **external documentation**.

movePageDown(This) -> boolean()

Types:

```
This = wxGrid()
```

See **external documentation**.

movePageUp(This) -> boolean()

Types:

```
This = wxGrid()
```

See **external documentation**.

registerDataType(This, TypeName, Renderer, Editor) -> ok

Types:

```
This = wxGrid()
TypeName = chardata() (see module unicode)
Renderer = wxGridCellRenderer() (see module wxGridCellRenderer)
Editor = wxGridCellEditor() (see module wxGridCellEditor)
```

See **external documentation**.

saveEditControlValue(This) -> ok

Types:

```
This = wxGrid()
```

See **external documentation**.

selectAll(This) -> ok

Types:

```
This = wxGrid()
```

See **external documentation**.

selectBlock(This, TopLeft, BottomRight) -> ok

Types:

```
This = wxGrid()
TopLeft = {R::integer(), C::integer()}
BottomRight = {R::integer(), C::integer()}
```

Equivalent to *selectBlock(This, TopLeft, BottomRight, [])*.

selectBlock(This, TopLeft, BottomRight, Option::[Option]) -> ok

Types:

```
This = wxGrid()
TopLeft = {R::integer(), C::integer()}
BottomRight = {R::integer(), C::integer()}
Option = {addToSelected, boolean()}
```

See **external documentation**.

selectBlock(This, TopRow, LeftCol, BottomRow, RightCol) -> ok

Types:

```
This = wxGrid()
TopRow = integer()
LeftCol = integer()
BottomRow = integer()
RightCol = integer()
```

Equivalent to *selectBlock(This, TopRow, LeftCol, BottomRow, RightCol, [])*.

selectBlock(This, TopRow, LeftCol, BottomRow, RightCol, Option::[Option]) -> ok

Types:

```
This = wxGrid()
TopRow = integer()
LeftCol = integer()
BottomRow = integer()
RightCol = integer()
Option = {addToSelected, boolean()}
```

See **external documentation**.

selectCol(This, Col) -> ok

Types:

```
This = wxGrid()
Col = integer()
```

Equivalent to *selectCol(This, Col, [])*.

selectCol(This, Col, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Col = integer()
Option = {addToSelected, boolean()}
```

See **external documentation**.

selectRow(This, Row) -> ok

Types:

```
This = wxGrid()
Row = integer()
```

Equivalent to *selectRow(This, Row, [])*.

selectRow(This, Row, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Row = integer()
Option = {addToSelected, boolean()}
```

See **external documentation**.

setCellAlignment(This, Align) -> ok

Types:

```
This = wxGrid()
Align = integer()
```

See **external documentation**.

setCellAlignment(This, Align, Row, Col) -> ok

Types:

```
This = wxGrid()
Align = integer()
Row = integer()
Col = integer()
```

See **external documentation**.

setCellAlignment(This, Row, Col, Horiz, Vert) -> ok

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
Horiz = integer()
Vert = integer()
```

See **external documentation**.

setCellBackgroundColour(This, Col) -> ok

Types:

```
This = wxGrid()
Col = wx_colour() (see module wx)
```

See **external documentation**.

setCellBackgroundColour(This, Row, Col, Val) -> ok

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
Val = wx_colour() (see module wx)
```

See **external documentation**.
Also:
setCellBackgroundColour(This, Colour, Row, Col) -> ok when
This::wxGrid(), Colour::wx:wx_colour(), Row::integer(), Col::integer().

setCellEditor(This, Row, Col, Editor) -> ok

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
Editor = wxGridCellEditor() (see module wxGridCellEditor)
```

See **external documentation**.

setCellFont(This, Row, Col, Val) -> ok

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
Val = wxFont() (see module wxFont)
```

See **external documentation**.

setCellRenderer(This, Row, Col, Renderer) -> ok

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
Renderer = wxGridCellRenderer() (see module wxGridCellRenderer)
```

See **external documentation**.

setCellTextColour(This, Col) -> ok

Types:

```
This = wxGrid()
Col = wx_colour() (see module wx)
```

See **external documentation**.

setCellTextColour(This, Row, Col, Val) -> ok

Types:

This = wxGrid()
Row = integer()
Col = integer()
Val = wx_colour() (see module wx)

See **external documentation**.
Also:
setCellTextColour(This, Val, Row, Col) -> ok when
This::wxGrid(), Val::wx:wx_colour(), Row::integer(), Col::integer().

setCellValue(This, Coords, S) -> ok

Types:

This = wxGrid()
Coords = {R::integer(), C::integer()}
S = chardata() (see module unicode)

See **external documentation**.

setCellValue(This, Row, Col, S) -> ok

Types:

This = wxGrid()
Row = integer()
Col = integer()
S = chardata() (see module unicode)

See **external documentation**.
Also:
setCellValue(This, Val, Row, Col) -> ok when
This::wxGrid(), Val::unicode:chardata(), Row::integer(), Col::integer().

setColAttr(This, Col, Attr) -> ok

Types:

This = wxGrid()
Col = integer()
Attr = wxGridCellAttr() (see module wxGridCellAttr)

See **external documentation**.

setColFormatBool(This, Col) -> ok

Types:

This = wxGrid()
Col = integer()

See **external documentation**.

setColFormatNumber(This, Col) -> ok

Types:

This = wxGrid()
Col = integer()

See **external documentation**.

setColFormatFloat(This, Col) -> ok

Types:

This = wxGrid()
Col = integer()

Equivalent to *setColFormatFloat(This, Col, [])*.

setColFormatFloat(This, Col, Option::[Option]) -> ok

Types:

This = wxGrid()
Col = integer()
Option = {width, integer()} | {precision, integer()}

See **external documentation**.

setColFormatCustom(This, Col, TypeName) -> ok

Types:

This = wxGrid()
Col = integer()
TypeName = chardata() (see module unicode)

See **external documentation**.

setColLabelAlignment(This, Horiz, Vert) -> ok

Types:

This = wxGrid()
Horiz = integer()
Vert = integer()

See **external documentation**.

setColLabelSize(This, Height) -> ok

Types:

This = wxGrid()
Height = integer()

See **external documentation**.

setColLabelValue(This, Col, Val) -> ok

Types:

This = wxGrid()
Col = integer()

```
Val = chardata() (see module unicode)
```

See **external documentation**.

```
setColMinimalWidth(This, Col, Width) -> ok
```

Types:

```
This = wxGrid()
Col = integer()
Width = integer()
```

See **external documentation**.

```
setColMinimalAcceptableWidth(This, Width) -> ok
```

Types:

```
This = wxGrid()
Width = integer()
```

See **external documentation**.

```
setColSize(This, Col, Width) -> ok
```

Types:

```
This = wxGrid()
Col = integer()
Width = integer()
```

See **external documentation**.

```
setDefaultCellAlignment(This, Horiz, Vert) -> ok
```

Types:

```
This = wxGrid()
Horiz = integer()
Vert = integer()
```

See **external documentation**.

```
setDefaultCellBackgroundColour(This, Val) -> ok
```

Types:

```
This = wxGrid()
Val = wx_colour() (see module wx)
```

See **external documentation**.

```
setDefaultCellFont(This, Val) -> ok
```

Types:

```
This = wxGrid()
Val = wxFont() (see module wxFont)
```

See **external documentation**.

setDefaultCellTextColour(This, Val) -> ok

Types:

```
This = wxGrid()
Val = wx_colour() (see module wx)
```

See **external documentation**.

setDefaultEditor(This, Editor) -> ok

Types:

```
This = wxGrid()
Editor = wxGridCellEditor() (see module wxGridCellEditor)
```

See **external documentation**.

setDefaultRenderer(This, Renderer) -> ok

Types:

```
This = wxGrid()
Renderer = wxGridCellRenderer() (see module wxGridCellRenderer)
```

See **external documentation**.

setDefaultColSize(This, Width) -> ok

Types:

```
This = wxGrid()
Width = integer()
```

Equivalent to *setDefaultColSize(This, Width, [])*.

setDefaultColSize(This, Width, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Width = integer()
Option = {resizeExistingCols, boolean()}
```

See **external documentation**.

setDefaultRowSize(This, Height) -> ok

Types:

```
This = wxGrid()
Height = integer()
```

Equivalent to *setDefaultRowSize(This, Height, [])*.

setDefaultRowSize(This, Height, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Height = integer()
Option = {resizeExistingRows, boolean()}
```

See **external documentation**.

setGridCursor(This, Row, Col) -> ok

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
```

See **external documentation**.

setGridLineColour(This, Val) -> ok

Types:

```
This = wxGrid()
Val = wx_colour() (see module wx)
```

See **external documentation**.

setLabelBackgroundColour(This, Val) -> ok

Types:

```
This = wxGrid()
Val = wx_colour() (see module wx)
```

See **external documentation**.

setLabelFont(This, Val) -> ok

Types:

```
This = wxGrid()
Val = wxFont() (see module wxFont)
```

See **external documentation**.

setLabelTextColour(This, Val) -> ok

Types:

```
This = wxGrid()
Val = wx_colour() (see module wx)
```

See **external documentation**.

setMargins(This, ExtraWidth, ExtraHeight) -> ok

Types:

```
This = wxGrid()
ExtraWidth = integer()
ExtraHeight = integer()
```

See **external documentation**.

setReadOnly(This, Row, Col) -> ok

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
```

Equivalent to *setReadOnly(This, Row, Col, [])*.

setReadOnly(This, Row, Col, Option::[Option]) -> ok

Types:

```
This = wxGrid()
Row = integer()
Col = integer()
Option = {isReadOnly, boolean()}
```

See **external documentation**.

setRowAttr(This, Row, Attr) -> ok

Types:

```
This = wxGrid()
Row = integer()
Attr = wxGridCellAttr() (see module wxGridCellAttr)
```

See **external documentation**.

setRowLabelAlignment(This, Horiz, Vert) -> ok

Types:

```
This = wxGrid()
Horiz = integer()
Vert = integer()
```

See **external documentation**.

setRowLabelSize(This, Width) -> ok

Types:

```
This = wxGrid()
Width = integer()
```

See **external documentation**.

setRowLabelValue(This, Row, Val) -> ok

Types:

```
This = wxGrid()
Row = integer()
Val = chardata() (see module unicode)
```

See **external documentation**.

setRowMinimalHeight(This, Row, Width) -> ok

Types:

```
This = wxGrid()
Row = integer()
Width = integer()
```

See **external documentation**.

setRowMinimalAcceptableHeight(This, Width) -> ok

Types:

```
This = wxGrid()
Width = integer()
```

See **external documentation**.

setRowSize(This, Row, Height) -> ok

Types:

```
This = wxGrid()
Row = integer()
Height = integer()
```

See **external documentation**.

setScrollLineX(This, X) -> ok

Types:

```
This = wxGrid()
X = integer()
```

See **external documentation**.

setScrollLineY(This, Y) -> ok

Types:

```
This = wxGrid()
Y = integer()
```

See **external documentation**.

setSelectionBackground(This, C) -> ok

Types:

```
This = wxGrid()
C = wx_colour() (see module wx)
```

See **external documentation**.

setSelectionForeground(This, C) -> ok

Types:

```
This = wxGrid()
C = wx_colour() (see module wx)
```

See **external documentation**.

setSelectionMode(This, Selmode) -> ok

Types:

```
This = wxGrid()
Selmode = wx_enum() (see module wx)
```

See **external documentation**.

Selmode = ?wxGrid_wxGridSelectCells | ?wxGrid_wxGridSelectRows | ?wxGrid_wxGridSelectColumns

showCellEditControl(This) -> ok

Types:

This = wxGrid()

See **external documentation**.

xToCol(This, X) -> integer()

Types:

This = wxGrid()

X = integer()

Equivalent to *xToCol(This, X, [])*.

xToCol(This, X, Option::[Option]) -> integer()

Types:

This = wxGrid()

X = integer()

Option = {clipToMinMax, boolean()}

See **external documentation**.

xToEdgeOfCol(This, X) -> integer()

Types:

This = wxGrid()

X = integer()

See **external documentation**.

yToEdgeOfRow(This, Y) -> integer()

Types:

This = wxGrid()

Y = integer()

See **external documentation**.

yToRow(This, Y) -> integer()

Types:

This = wxGrid()

Y = integer()

See **external documentation**.

destroy(This::wxGrid()) -> ok

Destroys this object, do not use object again

wxGridBagSizer

Erlang module

See external documentation: **wxGridBagSizer**.

This class is derived (and can use functions) from:
wxFlexGridSizer
wxGridSizer
wxSizer

DATA TYPES

`wxGridBagSizer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridBagSizer()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxGridBagSizer()`

Types:

```
Option = {vgap, integer()} | {hgap, integer()}
```

See **external documentation**.

`add(This, Item) -> wxSizerItem() (see module wxSizerItem)`

Types:

```
This = wxGridBagSizer()
Item = wxSizerItem() (see module wxSizerItem) | wxGBSizerItem() (see
module wxGBSizerItem)
```

See **external documentation**.

`add(This, Width, Height) -> wxSizerItem() (see module wxSizerItem)`

Types:

```
This = wxGridBagSizer()
Width = integer()
Height = integer()
```

See **external documentation**.
Also:
add(This, Window, Pos) -> wxSizerItem:wxSizerItem() when
This::wxGridBagSizer(), Window::wxWindow:wxWindow() | wxSizer:wxSizer(), Pos::{R::integer(), C::integer()};
(This, Window, [Option]) -> wxSizerItem:wxSizerItem() when
This::wxGridBagSizer(), Window::wxWindow:wxWindow() | wxSizer:wxSizer(),
Option :: {proportion, integer()}
| {flag, integer()}

| {border, integer()}
| {userData, wx:wx_object()}.

add(This, Width, Height, Pos) -> wxSizerItem() (see module wxSizerItem)

Types:

```
This = wxGridBagSizer()
Width = integer()
Height = integer()
Pos = {R::integer(), C::integer()}
```

See **external documentation**.
Also:
add(This, Width, Height, [Option]) -> wxSizerItem:wxSizerItem() when
This::wxGridBagSizer(), Width::integer(), Height::integer(),
Option :: {proportion, integer()}
| {flag, integer()}
| {border, integer()}
| {userData, wx:wx_object()};
(This, Window, Pos, [Option]) -> wxSizerItem:wxSizerItem() when
This::wxGridBagSizer(), Window::wxWindow:wxWindow() | wxSizer:wxSizer(), Pos::{R::integer(), C::integer()},
Option :: {span, {RS::integer(), CS::integer()}}
| {flag, integer()}
| {border, integer()}
| {userData, wx:wx_object()}.

add(This, Width, Height, Pos, Option::[Option]) -> wxSizerItem() (see module wxSizerItem)

Types:

```
This = wxGridBagSizer()
Width = integer()
Height = integer()
Pos = {R::integer(), C::integer()}
Option = {span, {RS::integer(), CS::integer()}} | {flag, integer()} |
{border, integer()} | {userData, wx_object() (see module wx)}
```

See **external documentation**.

calcMin(This) -> {W::integer(), H::integer()}

Types:

```
This = wxGridBagSizer()
```

See **external documentation**.

checkForIntersection(This, Item) -> boolean()

Types:

```
This = wxGridBagSizer()
Item = wxGBSizerItem() (see module wxGBSizerItem)
```

Equivalent to *checkForIntersection(This, Item, [])*.

checkForIntersection(This, Pos, Span) -> boolean()

Types:

This = wxGridBagSizer()
Pos = {R::integer(), C::integer()}
Span = {RS::integer(), CS::integer()}

See **external documentation**.
Also:
checkForIntersection(This, Item, [Option]) -> boolean() when
This::wxGridBagSizer(), Item::wxGBSizerItem:wxGBSizerItem(),
Option :: {excludeItem, wxGBSizerItem:wxGBSizerItem()}.

checkForIntersection(This, Pos, Span, Option::[Option]) -> boolean()

Types:

This = wxGridBagSizer()
Pos = {R::integer(), C::integer()}
Span = {RS::integer(), CS::integer()}
Option = {excludeItem, wxGBSizerItem() (see module wxGBSizerItem)}

See **external documentation**.

findItem(This, Window) -> wxGBSizerItem() (see module wxGBSizerItem)

Types:

This = wxGridBagSizer()
Window = wxWindow() (see module wxWindow) | wxSizer() (see module wxSizer)

See **external documentation**.

findItemAtPoint(This, Pt) -> wxGBSizerItem() (see module wxGBSizerItem)

Types:

This = wxGridBagSizer()
Pt = {X::integer(), Y::integer()}

See **external documentation**.

findItemAtPosition(This, Pos) -> wxGBSizerItem() (see module wxGBSizerItem)

Types:

This = wxGridBagSizer()
Pos = {R::integer(), C::integer()}

See **external documentation**.

findItemWithData(This, UserData) -> wxGBSizerItem() (see module wxGBSizerItem)

Types:

This = wxGridBagSizer()
UserData = wx_object() (see module wx)

See **external documentation**.

getCellSize(This, Row, Col) -> {W::integer(), H::integer()}

Types:

This = wxGridBagSizer()
Row = integer()
Col = integer()

See **external documentation**.

getEmptyCellSize(This) -> {W::integer(), H::integer()}

Types:

This = wxGridBagSizer()

See **external documentation**.

getItemPosition(This, Index) -> {R::integer(), C::integer()}

Types:

This = wxGridBagSizer()
Index = integer()

See **external documentation**.
Also:
getItemPosition(This, Window) -> {R::integer(), C::integer()} when
This::wxGridBagSizer(), Window::wxWindow:wxWindow() | wxSizer:wxSizer().

getItemSpan(This, Index) -> {RS::integer(), CS::integer()}

Types:

This = wxGridBagSizer()
Index = integer()

See **external documentation**.
Also:
getItemSpan(This, Window) -> {RS::integer(), CS::integer()} when
This::wxGridBagSizer(), Window::wxWindow:wxWindow() | wxSizer:wxSizer().

setEmptyCellSize(This, Sz) -> ok

Types:

This = wxGridBagSizer()
Sz = {W::integer(), H::integer()}

See **external documentation**.

setItemPosition(This, Index, Pos) -> boolean()

Types:

This = wxGridBagSizer()
Index = integer()
Pos = {R::integer(), C::integer()}

See **external documentation**.
Also:
setItemPosition(This, Window, Pos) -> boolean() when

This::wxGridBagSizer(), Window::wxWindow:wxWindow() | wxSizer:wxSizer(), Pos::{R::integer(), C::integer()}.

setItemSpan(This, Index, Span) -> boolean()

Types:

This = wxGridBagSizer()
Index = integer()
Span = {RS::integer(), CS::integer()}

See **external documentation**.
Also:
setItemSpan(This, Window, Span) -> boolean() when This::wxGridBagSizer(), Window::wxWindow:wxWindow() | wxSizer:wxSizer(), Span::{RS::integer(), CS::integer()}.

destroy(This::wxGridBagSizer()) -> ok

Destroys this object, do not use object again

wxGridCellAttr

Erlang module

See external documentation: **wxGridCellAttr**.

DATA TYPES

`wxGridCellAttr()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`setTextColour(This, ColText) -> ok`

Types:

```
This = wxGridCellAttr()
ColText = wx_colour() (see module wx)
```

See **external documentation**.

`setBackgroundColour(This, ColBack) -> ok`

Types:

```
This = wxGridCellAttr()
ColBack = wx_colour() (see module wx)
```

See **external documentation**.

`setFont(This, Font) -> ok`

Types:

```
This = wxGridCellAttr()
Font = wxFont() (see module wxFont)
```

See **external documentation**.

`setAlignment(This, HAlign, VAlign) -> ok`

Types:

```
This = wxGridCellAttr()
HAlign = integer()
VAlign = integer()
```

See **external documentation**.

`setReadOnly(This) -> ok`

Types:

```
This = wxGridCellAttr()
```

Equivalent to *setReadOnly(This, [])*.

setReadOnly(This, Option::[Option]) -> ok

Types:

This = wxGridCellAttr()
Option = {isReadOnly, boolean()}

See **external documentation**.

setRenderer(This, Renderer) -> ok

Types:

This = wxGridCellAttr()
Renderer = wxGridCellRenderer() (see module wxGridCellRenderer)

See **external documentation**.

setEditor(This, Editor) -> ok

Types:

This = wxGridCellAttr()
Editor = wxGridCellEditor() (see module wxGridCellEditor)

See **external documentation**.

hasTextColour(This) -> boolean()

Types:

This = wxGridCellAttr()

See **external documentation**.

hasBackgroundColour(This) -> boolean()

Types:

This = wxGridCellAttr()

See **external documentation**.

hasFont(This) -> boolean()

Types:

This = wxGridCellAttr()

See **external documentation**.

hasAlignment(This) -> boolean()

Types:

This = wxGridCellAttr()

See **external documentation**.

hasRenderer(This) -> boolean()

Types:

This = wxGridCellAttr()

See **external documentation**.

hasEditor(This) -> boolean()

Types:

This = wxGridCellAttr()

See **external documentation**.

getTextColour(This) -> wx_colour4() (see module wx)

Types:

This = wxGridCellAttr()

See **external documentation**.

getBackgroundColour(This) -> wx_colour4() (see module wx)

Types:

This = wxGridCellAttr()

See **external documentation**.

getFont(This) -> wxFont() (see module wxFont)

Types:

This = wxGridCellAttr()

See **external documentation**.

getAlignment(This) -> {HAlign::integer(), VAlign::integer()}

Types:

This = wxGridCellAttr()

See **external documentation**.

getRenderer(This, Grid, Row, Col) -> wxGridCellRenderer() (see module wxGridCellRenderer)

Types:

This = wxGridCellAttr()

Grid = wxGrid() (see module wxGrid)

Row = integer()

Col = integer()

See **external documentation**.

getEditor(This, Grid, Row, Col) -> wxGridCellEditor() (see module wxGridCellEditor)

Types:

This = wxGridCellAttr()

Grid = wxGrid() (see module wxGrid)

Row = integer()

Col = integer()

See **external documentation**.

isReadOnly(This) -> boolean()

Types:

This = wxGridCellAttr()

See **external documentation**.

setDefAttr(This, DefAttr) -> ok

Types:

This = wxGridCellAttr()

DefAttr = wxGridCellAttr()

See **external documentation**.

wxGridCellBoolEditor

Erlang module

See external documentation: **wxGridCellBoolEditor**.

This class is derived (and can use functions) from:
wxGridCellEditor

DATA TYPES

`wxGridCellBoolEditor()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridCellBoolEditor()`

See **external documentation**.

`isTrueValue(Value) -> boolean()`

Types:

`Value = chardata() (see module unicode)`

See **external documentation**.

`useStringValues() -> ok`

Equivalent to *useStringValues([])*.

`useStringValues(Option::[Option]) -> ok`

Types:

`Option = {valueTrue, chardata() (see module unicode)} | {valueFalse, chardata() (see module unicode)}`

See **external documentation**.

`destroy(This::wxGridCellBoolEditor()) -> ok`

Destroys this object, do not use object again

wxGridCellBoolRenderer

Erlang module

See external documentation: **wxGridCellBoolRenderer**.

This class is derived (and can use functions) from:
wxGridCellRenderer

DATA TYPES

`wxGridCellBoolRenderer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridCellBoolRenderer()`

See **external documentation**.

`destroy(This::wxGridCellBoolRenderer()) -> ok`

Destroys this object, do not use object again

wxGridCellChoiceEditor

Erlang module

See external documentation: **wxGridCellChoiceEditor**.

This class is derived (and can use functions) from:
wxGridCellEditor

DATA TYPES

`wxGridCellChoiceEditor()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Choices) -> wxGridCellChoiceEditor()`

Types:

```
Choices = [chardata() (see module unicode)]
```

Equivalent to *new(Choices, [])*.

`new(Choices, Option::[Option]) -> wxGridCellChoiceEditor()`

Types:

```
Choices = [chardata() (see module unicode)]
Option = {allowOthers, boolean()}
```

See **external documentation**.

`setParameters(This, Params) -> ok`

Types:

```
This = wxGridCellChoiceEditor()
Params = chardata() (see module unicode)
```

See **external documentation**.

`destroy(This::wxGridCellChoiceEditor()) -> ok`

Destroys this object, do not use object again

wxGridCellEditor

Erlang module

See external documentation: **wxGridCellEditor**.

DATA TYPES

`wxGridCellEditor()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`create(This, Parent, Id, EvtHandler) -> ok`

Types:

```
This = wxGridCellEditor()
Parent = wxWindow() (see module wxWindow)
Id = integer()
EvtHandler = wxEvtHandler() (see module wxEvtHandler)
```

See **external documentation**.

`isCreated(This) -> boolean()`

Types:

```
This = wxGridCellEditor()
```

See **external documentation**.

`setSize(This, Rect) -> ok`

Types:

```
This = wxGridCellEditor()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

See **external documentation**.

`show(This, Show) -> ok`

Types:

```
This = wxGridCellEditor()
Show = boolean()
```

Equivalent to *show(This, Show, [])*.

`show(This, Show, Option::[Option]) -> ok`

Types:

```
This = wxGridCellEditor()
Show = boolean()
Option = {attr, wxGridCellAttr() (see module wxGridCellAttr)}
```

See **external documentation**.

paintBackground(This, RectCell, Attr) -> ok

Types:

```
This = wxGridCellEditor()
RectCell = {X::integer(), Y::integer(), W::integer(), H::integer()}
Attr = wxGridCellAttr() (see module wxGridCellAttr)
```

See **external documentation**.

beginEdit(This, Row, Col, Grid) -> ok

Types:

```
This = wxGridCellEditor()
Row = integer()
Col = integer()
Grid = wxGrid() (see module wxGrid)
```

See **external documentation**.

endEdit(This, Row, Col, Grid) -> boolean()

Types:

```
This = wxGridCellEditor()
Row = integer()
Col = integer()
Grid = wxGrid() (see module wxGrid)
```

See **external documentation**.

reset(This) -> ok

Types:

```
This = wxGridCellEditor()
```

See **external documentation**.

startingKey(This, Event) -> ok

Types:

```
This = wxGridCellEditor()
Event = wxKeyEvent() (see module wxKeyEvent)
```

See **external documentation**.

startingClick(This) -> ok

Types:

```
This = wxGridCellEditor()
```

See **external documentation**.

handleReturn(This, Event) -> ok

Types:

```
This = wxGridCellEditor()
Event = wxKeyEvent() (see module wxKeyEvent)
```

See **external documentation**.

wxGridCellFloatEditor

Erlang module

See external documentation: **wxGridCellFloatEditor**.

This class is derived (and can use functions) from:
wxGridCellEditor

DATA TYPES

`wxGridCellFloatEditor()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridCellFloatEditor()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxGridCellFloatEditor()`

Types:

```
Option = {width, integer()} | {precision, integer()}
```

See **external documentation**.

`setParameters(This, Params) -> ok`

Types:

```
This = wxGridCellFloatEditor()
Params = chardata() (see module unicode)
```

See **external documentation**.

`destroy(This::wxGridCellFloatEditor()) -> ok`

Destroys this object, do not use object again

wxGridCellFloatRenderer

Erlang module

See external documentation: **wxGridCellFloatRenderer**.

This class is derived (and can use functions) from:
wxGridCellStringRenderer
wxGridCellRenderer

DATA TYPES

`wxGridCellFloatRenderer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridCellFloatRenderer()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxGridCellFloatRenderer()`

Types:

```
Option = {width, integer()} | {precision, integer()}
```

See **external documentation**.

`getPrecision(This) -> integer()`

Types:

```
This = wxGridCellFloatRenderer()
```

See **external documentation**.

`getWidth(This) -> integer()`

Types:

```
This = wxGridCellFloatRenderer()
```

See **external documentation**.

`setParameters(This, Params) -> ok`

Types:

```
This = wxGridCellFloatRenderer()
Params = chardata() (see module unicode)
```

See **external documentation**.

`setPrecision(This, Precision) -> ok`

Types:

```
This = wxGridCellFloatRenderer()
```

```
Precision = integer()
```

See **external documentation**.

setWidth(This, Width) -> ok

Types:

```
This = wxGridCellFloatRenderer()
Width = integer()
```

See **external documentation**.

destroy(This::wxGridCellFloatRenderer()) -> ok

Destroys this object, do not use object again

wxGridCellNumberEditor

Erlang module

See external documentation: **wxGridCellNumberEditor**.

This class is derived (and can use functions) from:
wxGridCellTextEditor
wxGridCellEditor

DATA TYPES

`wxGridCellNumberEditor()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridCellNumberEditor()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxGridCellNumberEditor()`

Types:

```
Option = {min, integer()} | {max, integer()}
```

See **external documentation**.

`getValue(This) -> charlist() (see module unicode)`

Types:

```
This = wxGridCellNumberEditor()
```

See **external documentation**.

`setParameters(This, Params) -> ok`

Types:

```
This = wxGridCellNumberEditor()
Params = chardata() (see module unicode)
```

See **external documentation**.

`destroy(This::wxGridCellNumberEditor()) -> ok`

Destroys this object, do not use object again

wxGridCellNumberRenderer

Erlang module

See external documentation: **wxGridCellNumberRenderer**.

This class is derived (and can use functions) from:
wxGridCellStringRenderer
wxGridCellRenderer

DATA TYPES

`wxGridCellNumberRenderer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridCellNumberRenderer()`

See **external documentation**.

`destroy(This::wxGridCellNumberRenderer()) -> ok`

Destroys this object, do not use object again

wxGridCellRenderer

Erlang module

See external documentation: **wxGridCellRenderer**.

DATA TYPES

`wxGridCellRenderer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`draw(This, Grid, Attr, Dc, Rect, Row, Col, IsSelected) -> ok`

Types:

```
This = wxGridCellRenderer()
Grid = wxGrid() (see module wxGrid)
Attr = wxGridCellAttr() (see module wxGridCellAttr)
Dc = wxDC() (see module wxDC)
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
Row = integer()
Col = integer()
IsSelected = boolean()
```

See **external documentation**.

`getBestSize(This, Grid, Attr, Dc, Row, Col) -> {W::integer(), H::integer()}`

Types:

```
This = wxGridCellRenderer()
Grid = wxGrid() (see module wxGrid)
Attr = wxGridCellAttr() (see module wxGridCellAttr)
Dc = wxDC() (see module wxDC)
Row = integer()
Col = integer()
```

See **external documentation**.

wxGridCellStringRenderer

Erlang module

See external documentation: **wxGridCellStringRenderer**.

This class is derived (and can use functions) from:
wxGridCellRenderer

DATA TYPES

`wxGridCellStringRenderer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridCellStringRenderer()`

See **external documentation**.

`destroy(This::wxGridCellStringRenderer()) -> ok`

Destroys this object, do not use object again

wxGridCellTextEditor

Erlang module

See external documentation: **wxGridCellTextEditor**.

This class is derived (and can use functions) from:
wxGridCellEditor

DATA TYPES

`wxGridCellTextEditor()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxGridCellTextEditor()`

See **external documentation**.

`setParameters(This, Params) -> ok`

Types:

`This = wxGridCellTextEditor()`
`Params = chardata() (see module unicode)`

See **external documentation**.

`destroy(This::wxGridCellTextEditor()) -> ok`

Destroys this object, do not use object again

wxGridEvent

Erlang module

See external documentation: **wxGridEvent**.

Use *wxEvtHandler:connect/3* with EventType:

grid_cell_left_click, *grid_cell_right_click*, *grid_cell_left_dclick*, *grid_cell_right_dclick*, *grid_label_left_click*, *grid_label_right_click*, *grid_label_left_dclick*, *grid_label_right_dclick*, *grid_row_size*, *grid_col_size*, *grid_range_select*, *grid_cell_change*, *grid_select_cell*, *grid_editor_shown*, *grid_editor_hidden*, *grid_editor_created*, *grid_cell_begin_drag*

See also the message variant *#wxGrid{}* event record type.

This class is derived (and can use functions) from:
wxNotifyEvent
wxCommandEvent
wxEvent

DATA TYPES

`wxGridEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`altDown(This) -> boolean()`

Types:

`This = wxGridEvent()`

See **external documentation**.

`controlDown(This) -> boolean()`

Types:

`This = wxGridEvent()`

See **external documentation**.

`getCol(This) -> integer()`

Types:

`This = wxGridEvent()`

See **external documentation**.

`getPosition(This) -> {X::integer(), Y::integer()}`

Types:

`This = wxGridEvent()`

See **external documentation**.

getRow(This) -> integer()

Types:

This = wxGridEvent()

See **external documentation**.

metaDown(This) -> boolean()

Types:

This = wxGridEvent()

See **external documentation**.

selecting(This) -> boolean()

Types:

This = wxGridEvent()

See **external documentation**.

shiftDown(This) -> boolean()

Types:

This = wxGridEvent()

See **external documentation**.

wxGridSizer

Erlang module

See external documentation: **wxGridSizer**.

This class is derived (and can use functions) from:
wxSizer

DATA TYPES

`wxGridSizer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Cols) -> wxGridSizer()`

Types:

```
Cols = integer()
```

Equivalent to *new(Cols, [])*.

`new(Cols, Option::[Option]) -> wxGridSizer()`

Types:

```
Cols = integer()
Option = {vgap, integer()} | {hgap, integer()}
```

See **external documentation**.

`new(Rows, Cols, Vgap, Hgap) -> wxGridSizer()`

Types:

```
Rows = integer()
Cols = integer()
Vgap = integer()
Hgap = integer()
```

See **external documentation**.

`getCols(This) -> integer()`

Types:

```
This = wxGridSizer()
```

See **external documentation**.

`getHGap(This) -> integer()`

Types:

```
This = wxGridSizer()
```

See **external documentation**.

getRows(This) -> integer()

Types:

This = wxGridSizer()

See **external documentation**.

getVGap(This) -> integer()

Types:

This = wxGridSizer()

See **external documentation**.

setCols(This, Cols) -> ok

Types:

This = wxGridSizer()

Cols = integer()

See **external documentation**.

setHGap(This, Gap) -> ok

Types:

This = wxGridSizer()

Gap = integer()

See **external documentation**.

setRows(This, Rows) -> ok

Types:

This = wxGridSizer()

Rows = integer()

See **external documentation**.

setVGap(This, Gap) -> ok

Types:

This = wxGridSizer()

Gap = integer()

See **external documentation**.

destroy(This::wxGridSizer()) -> ok

Destroys this object, do not use object again

wxHelpEvent

Erlang module

See external documentation: **wxHelpEvent**.

Use *wxEvtHandler:connect/3* with EventType:

help, detailed_help

See also the message variant *#wxHelp{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxHelpEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getOrigin(This) -> wx_enum() (see module wx)`

Types:

`This = wxHelpEvent()`

See **external documentation**.
Res = ?wxHelpEvent_Origin_Unknown | ?wxHelpEvent_Origin_Keyboard | ?wxHelpEvent_Origin_HelpButton

`getPosition(This) -> {X::integer(), Y::integer()}`

Types:

`This = wxHelpEvent()`

See **external documentation**.

`setOrigin(This, Origin) -> ok`

Types:

`This = wxHelpEvent()`
`Origin = wx_enum() (see module wx)`

See **external documentation**.
Origin = ?wxHelpEvent_Origin_Unknown | ?wxHelpEvent_Origin_Keyboard | ?wxHelpEvent_Origin_HelpButton

`setPosition(This, Pos) -> ok`

Types:

`This = wxHelpEvent()`
`Pos = {X::integer(), Y::integer()}`

See **external documentation**.

wxHtmlEasyPrinting

Erlang module

See external documentation: **wxHtmlEasyPrinting**.

DATA TYPES

`wxHtmlEasyPrinting()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxHtmlEasyPrinting()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxHtmlEasyPrinting()`

Types:

```
Option = {name, chardata() (see module unicode)} | {parentWindow,
wxWindow() (see module wxWindow)}
```

See **external documentation**.

`getPrintData(This) -> wxPrintData() (see module wxPrintData)`

Types:

```
This = wxHtmlEasyPrinting()
```

See **external documentation**.

`getPageSetupData(This) -> wxPageSetupDialogData() (see module wxPageSetupDialogData)`

Types:

```
This = wxHtmlEasyPrinting()
```

See **external documentation**.

`previewFile(This, Htmlfile) -> boolean()`

Types:

```
This = wxHtmlEasyPrinting()
Htmlfile = chardata() (see module unicode)
```

See **external documentation**.

`previewText(This, Htmltext) -> boolean()`

Types:

```
This = wxHtmlEasyPrinting()
Htmltext = chardata() (see module unicode)
```

Equivalent to *previewText(This, Htmltext, [])*.

previewText(This, Htmltext, Option::[Option]) -> boolean()

Types:

```
This = wxHtmlEasyPrinting()
Htmltext = chardata() (see module unicode)
Option = {basepath, chardata() (see module unicode)}
```

See **external documentation**.

printFile(This, Htmlfile) -> boolean()

Types:

```
This = wxHtmlEasyPrinting()
Htmlfile = chardata() (see module unicode)
```

See **external documentation**.

printText(This, Htmltext) -> boolean()

Types:

```
This = wxHtmlEasyPrinting()
Htmltext = chardata() (see module unicode)
```

Equivalent to *printText(This, Htmltext, [])*.

printText(This, Htmltext, Option::[Option]) -> boolean()

Types:

```
This = wxHtmlEasyPrinting()
Htmltext = chardata() (see module unicode)
Option = {basepath, chardata() (see module unicode)}
```

See **external documentation**.

pageSetup(This) -> ok

Types:

```
This = wxHtmlEasyPrinting()
```

See **external documentation**.

setFonts(This, Normal_face, Fixed_face) -> ok

Types:

```
This = wxHtmlEasyPrinting()
Normal_face = chardata() (see module unicode)
Fixed_face = chardata() (see module unicode)
```

Equivalent to *setFonts(This, Normal_face, Fixed_face, [])*.

setFonts(This, Normal_face, Fixed_face, Option::[Option]) -> ok

Types:

```
This = wxHtmlEasyPrinting()
```

```
Normal_face = chardata() (see module unicode)
Fixed_face = chardata() (see module unicode)
Option = {sizes, [integer()]}
```

See **external documentation**.

setHeader(This, Header) -> ok

Types:

```
This = wxHtmlEasyPrinting()
Header = chardata() (see module unicode)
```

Equivalent to *setHeader(This, Header, [])*.

setHeader(This, Header, Option::[Option]) -> ok

Types:

```
This = wxHtmlEasyPrinting()
Header = chardata() (see module unicode)
Option = {pg, integer()}
```

See **external documentation**.

setFooter(This, Footer) -> ok

Types:

```
This = wxHtmlEasyPrinting()
Footer = chardata() (see module unicode)
```

Equivalent to *setFooter(This, Footer, [])*.

setFooter(This, Footer, Option::[Option]) -> ok

Types:

```
This = wxHtmlEasyPrinting()
Footer = chardata() (see module unicode)
Option = {pg, integer()}
```

See **external documentation**.

destroy(This::wxHtmlEasyPrinting()) -> ok

Destroys this object, do not use object again

wxHtmlLinkEvent

Erlang module

See external documentation: **wxHtmlLinkEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_html_link_clicked

See also the message variant *#wxHtmlLink{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxHtmlLinkEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getLinkInfo(This) -> wx_wxHtmlLinkInfo() (see module wx)`

Types:

`This = wxHtmlLinkEvent()`

See **external documentation**.

wxHtmlWindow

Erlang module

See external documentation: **wxHtmlWindow**.

This class is derived (and can use functions) from:
wxScrolledWindow
wxPanel
wxWindow
wxEvtHandler

DATA TYPES

`wxHtmlWindow()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxHtmlWindow()`

See **external documentation**.

`new(Parent) -> wxHtmlWindow()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxHtmlWindow()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`appendToPage(This, Source) -> boolean()`

Types:

```
This = wxHtmlWindow()
Source = chardata() (see module unicode)
```

See **external documentation**.

`getOpenedAnchor(This) -> charlist() (see module unicode)`

Types:

```
This = wxHtmlWindow()
```

See **external documentation**.

getOpenedPage(This) -> charlist() (see module unicode)

Types:

This = wxHtmlWindow()

See **external documentation**.

getOpenedPageTitle(This) -> charlist() (see module unicode)

Types:

This = wxHtmlWindow()

See **external documentation**.

getRelatedFrame(This) -> wxFrame() (see module wxFrame)

Types:

This = wxHtmlWindow()

See **external documentation**.

historyBack(This) -> boolean()

Types:

This = wxHtmlWindow()

See **external documentation**.

historyCanBack(This) -> boolean()

Types:

This = wxHtmlWindow()

See **external documentation**.

historyCanForward(This) -> boolean()

Types:

This = wxHtmlWindow()

See **external documentation**.

historyClear(This) -> ok

Types:

This = wxHtmlWindow()

See **external documentation**.

historyForward(This) -> boolean()

Types:

This = wxHtmlWindow()

See **external documentation**.

loadFile(This, Filename) -> boolean()

Types:

This = wxHtmlWindow()

```
Filename = chardata() (see module unicode)
```

See **external documentation**.

```
loadPage(This, Location) -> boolean()
```

Types:

```
This = wxHtmlWindow()
Location = chardata() (see module unicode)
```

See **external documentation**.

```
selectAll(This) -> ok
```

Types:

```
This = wxHtmlWindow()
```

See **external documentation**.

```
selectionToText(This) -> charlist() (see module unicode)
```

Types:

```
This = wxHtmlWindow()
```

See **external documentation**.

```
selectLine(This, Pos) -> ok
```

Types:

```
This = wxHtmlWindow()
Pos = {X::integer(), Y::integer()}
```

See **external documentation**.

```
selectWord(This, Pos) -> ok
```

Types:

```
This = wxHtmlWindow()
Pos = {X::integer(), Y::integer()}
```

See **external documentation**.

```
setBorders(This, B) -> ok
```

Types:

```
This = wxHtmlWindow()
B = integer()
```

See **external documentation**.

```
setFonts(This, Normal_face, Fixed_face) -> ok
```

Types:

```
This = wxHtmlWindow()
Normal_face = chardata() (see module unicode)
Fixed_face = chardata() (see module unicode)
```

Equivalent to *setFonts(This, Normal_face, Fixed_face, [])*.

setFonts(This, Normal_face, Fixed_face, Option::[Option]) -> ok

Types:

This = wxHtmlWindow()
Normal_face = chardata() (see module unicode)
Fixed_face = chardata() (see module unicode)
Option = {sizes, integer()}

See **external documentation**.

setPage(This, Source) -> boolean()

Types:

This = wxHtmlWindow()
Source = chardata() (see module unicode)

See **external documentation**.

setRelatedFrame(This, Frame, Format) -> ok

Types:

This = wxHtmlWindow()
Frame = wxFrame() (see module wxFrame)
Format = chardata() (see module unicode)

See **external documentation**.

setRelatedStatusBar(This, Bar) -> ok

Types:

This = wxHtmlWindow()
Bar = integer()

See **external documentation**.

toText(This) -> charlist() (see module unicode)

Types:

This = wxHtmlWindow()

See **external documentation**.

destroy(This::wxHtmlWindow()) -> ok

Destroys this object, do not use object again

wxIcon

Erlang module

See external documentation: **wxIcon**.

This class is derived (and can use functions) from:
wxBitmap

DATA TYPES

`wxIcon()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxIcon()`

See **external documentation**.

`new(Filename) -> wxIcon()`

Types:

`Filename = chardata() (see module unicode)`

See **external documentation**.
Also:
new(Loc) -> wxIcon() when
Loc::wx:wx_object().

Type = ?wxBITMAP_TYPE_INVALID | ?wxBITMAP_TYPE_BMP | ?wxBITMAP_TYPE_BMP_RESOURCE | ?wxBITMAP_TYPE_RESOURCE | ?wxBITMAP_TYPE_ICO | ?wxBITMAP_TYPE_ICO_RESOURCE | ?wxBITMAP_TYPE_CUR | ?wxBITMAP_TYPE_CUR_RESOURCE | ?wxBITMAP_TYPE_XBM | ?wxBITMAP_TYPE_XBM_DATA | ?wxBITMAP_TYPE_XPM | ?wxBITMAP_TYPE_XPM_DATA | ?wxBITMAP_TYPE_TIF | ?wxBITMAP_TYPE_TIF_RESOURCE | ?wxBITMAP_TYPE_GIF | ?wxBITMAP_TYPE_GIF_RESOURCE | ?wxBITMAP_TYPE_PNG | ?wxBITMAP_TYPE_PNG_RESOURCE | ?wxBITMAP_TYPE_JPEG | ?wxBITMAP_TYPE_JPEG_RESOURCE | ?wxBITMAP_TYPE_PNM | ?wxBITMAP_TYPE_PNM_RESOURCE | ?wxBITMAP_TYPE_PCX | ?wxBITMAP_TYPE_PCX_RESOURCE | ?wxBITMAP_TYPE_PICT | ?wxBITMAP_TYPE_PICT_RESOURCE | ?wxBITMAP_TYPE_ICON | ?wxBITMAP_TYPE_ICON_RESOURCE | ?wxBITMAP_TYPE_ANI | ?wxBITMAP_TYPE_IFF | ?wxBITMAP_TYPE_TGA | ?wxBITMAP_TYPE_MACCURSOR | ?wxBITMAP_TYPE_MACCURSOR_RESOURCE | ?wxBITMAP_TYPE_ANY

`new(Filename, Option::[Option]) -> wxIcon()`

Types:

`Filename = chardata() (see module unicode)`

`Option = {type, wx_enum() (see module wx)} | {desiredWidth, integer()} | {desiredHeight, integer()}`

See **external documentation**.
Type = ?wxBITMAP_TYPE_INVALID | ?wxBITMAP_TYPE_BMP | ?wxBITMAP_TYPE_BMP_RESOURCE | ?wxBITMAP_TYPE_RESOURCE | ?wxBITMAP_TYPE_ICO | ?wxBITMAP_TYPE_ICO_RESOURCE

| ?wxBITMAP_TYPE_CUR | ?wxBITMAP_TYPE_CUR_RESOURCE | ?wxBITMAP_TYPE_XBM | ?wxBITMAP_TYPE_XBM_DATA | ?wxBITMAP_TYPE_XPM | ?wxBITMAP_TYPE_XPM_DATA | ?wxBITMAP_TYPE_TIF | ?wxBITMAP_TYPE_TIF_RESOURCE | ?wxBITMAP_TYPE_GIF | ?wxBITMAP_TYPE_GIF_RESOURCE | ?wxBITMAP_TYPE_PNG | ?wxBITMAP_TYPE_PNG_RESOURCE | ?wxBITMAP_TYPE_JPEG | ?wxBITMAP_TYPE_JPEG_RESOURCE | ?wxBITMAP_TYPE_PNM | ?wxBITMAP_TYPE_PNM_RESOURCE | ?wxBITMAP_TYPE_PCX | ?wxBITMAP_TYPE_PCX_RESOURCE | ?wxBITMAP_TYPE_PICT | ?wxBITMAP_TYPE_PICT_RESOURCE | ?wxBITMAP_TYPE_ICON | ?wxBITMAP_TYPE_ICON_RESOURCE | ?wxBITMAP_TYPE_ANI | ?wxBITMAP_TYPE_IFF | ?wxBITMAP_TYPE_TGA | ?wxBITMAP_TYPE_MACCURSOR | ?wxBITMAP_TYPE_MACCURSOR_RESOURCE | ?wxBITMAP_TYPE_ANY

`copyFromBitmap(This, Bmp) -> ok`

Types:

```
This = wxIcon()
Bmp = wxBitmap() (see module wxBitmap)
```

See **external documentation**.

`destroy(This::wxIcon()) -> ok`

Destroys this object, do not use object again

wxIconBundle

Erlang module

See external documentation: **wxIconBundle**.

DATA TYPES

`wxIconBundle()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxIconBundle()`

See **external documentation**.

`new(Ic) -> wxIconBundle()`

Types:

`Ic = wxIconBundle() | wxIcon() (see module wxIcon)`

See **external documentation**.

`new(File, Type) -> wxIconBundle()`

Types:

`File = chardata() (see module unicode)`
`Type = integer()`

See **external documentation**.

`addIcon(This, Icon) -> ok`

Types:

`This = wxIconBundle()`
`Icon = wxIcon() (see module wxIcon)`

See **external documentation**.

`addIcon(This, File, Type) -> ok`

Types:

`This = wxIconBundle()`
`File = chardata() (see module unicode)`
`Type = integer()`

See **external documentation**.

`getIcon(This) -> wxIcon() (see module wxIcon)`

Types:

`This = wxIconBundle()`

Equivalent to *getIcon(This, [])*.

`getIcon(This, Option::[Option]) -> wxIcon() (see module wxIcon)`

Types:

```
This = wxIconBundle()
Option = {size, integer()}
```

See **external documentation**.
Also:
getIcon(This, Size) -> wxIcon:wxIcon() when
This::wxIconBundle(), Size::{W::integer(), H::integer()}.

`destroy(This::wxIconBundle()) -> ok`

Destroys this object, do not use object again

wxIconizeEvent

Erlang module

See external documentation: **wxIconizeEvent**.

Use *wxEvtHandler:connect/3* with EventType:

iconize

See also the message variant *#wxIconize{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxIconizeEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`iconized(This) -> boolean()`

Types:

`This = wxIconizeEvent()`

See **external documentation**.

wxIdleEvent

Erlang module

See external documentation: **wxIdleEvent**.

Use *wxEvtHandler:connect/3* with EventType:

idle

See also the message variant *#wxIdle{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxIdleEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`canSend(Win) -> boolean()`

Types:

`Win = wxWindow() (see module wxWindow)`

See **external documentation**.

`getMode() -> wx_enum() (see module wx)`

See **external documentation**.
Res = ?wxIDLE_PROCESS_ALL | ?wxIDLE_PROCESS_SPECIFIED

`requestMore(This) -> ok`

Types:

`This = wxIdleEvent()`

Equivalent to *requestMore(This, [])*.

`requestMore(This, Option::[Option]) -> ok`

Types:

`This = wxIdleEvent()`
`Option = {needMore, boolean()}`

See **external documentation**.

`moreRequested(This) -> boolean()`

Types:

`This = wxIdleEvent()`

See **external documentation**.

setMode(Mode) -> ok

Types:

Mode = wx_enum() (see module wx)

See **external documentation**.
Mode = ?wxIDLE_PROCESS_ALL | ?wxIDLE_PROCESS_SPECIFIED

wxImage

Erlang module

See external documentation: **wxImage**.

All (default) image handlers are initialized.

DATA TYPES

`wxImage()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxImage()`

See **external documentation**.

`new(Name) -> wxImage()`

Types:

`Name = chardata() (see module unicode)`

Equivalent to *new(Name, [])*.

`new(Width, Height) -> wxImage()`

Types:

`Width = integer()`

`Height = integer()`

See **external documentation**.
Also:
new(Name, [Option]) -> wxImage() when
Name::unicode:chardata(),
Option :: {type, integer()}
| {index, integer()}.

`new(Width, Height, Data) -> wxImage()`

Types:

`Width = integer()`

`Height = integer()`

`Data = binary()`

See **external documentation**.
Also:
new(Width, Height, [Option]) -> wxImage() when
Width::integer(), Height::integer(),
Option :: {clear, boolean()};
(Name, Mimetype, [Option]) -> wxImage() when
Name::unicode:chardata(), Mimetype::unicode:chardata(),

Option :: {index, integer()}.

new(Width, Height, Data, Alpha) -> wxImage()

Types:

```
Width = integer()
Height = integer()
Data = binary()
Alpha = binary()
```

See **external documentation**.
Also:
new(Width, Height, Data, [Option]) -> wxImage() when
Width::integer(), Height::integer(), Data::binary(),
Option :: {static_data, boolean()}.

new(Width, Height, Data, Alpha, Option::[Option]) -> wxImage()

Types:

```
Width = integer()
Height = integer()
Data = binary()
Alpha = binary()
Option = {static_data, boolean()}
```

See **external documentation**.

blur(This, Radius) -> wxImage()

Types:

```
This = wxImage()
Radius = integer()
```

See **external documentation**.

blurHorizontal(This, Radius) -> wxImage()

Types:

```
This = wxImage()
Radius = integer()
```

See **external documentation**.

blurVertical(This, Radius) -> wxImage()

Types:

```
This = wxImage()
Radius = integer()
```

See **external documentation**.

convertAlphaToMask(This) -> boolean()

Types:

```
This = wxImage()
```

Equivalent to *convertAlphaToMask(This, [])*.

convertAlphaToMask(This, Option::[Option]) -> boolean()

Types:

This = wxImage()
Option = {threshold, integer()}

See **external documentation**.

convertToGreyscale(This) -> wxImage()

Types:

This = wxImage()

Equivalent to *convertToGreyscale(This, [])*.

convertToGreyscale(This, Option::[Option]) -> wxImage()

Types:

This = wxImage()
Option = {lr, number()} | {lg, number()} | {lb, number()}

See **external documentation**.

convertToMono(This, R, G, B) -> wxImage()

Types:

This = wxImage()
R = integer()
G = integer()
B = integer()

See **external documentation**.

copy(This) -> wxImage()

Types:

This = wxImage()

See **external documentation**.

create(This, Width, Height) -> boolean()

Types:

This = wxImage()
Width = integer()
Height = integer()

Equivalent to *create(This, Width, Height, [])*.

create(This, Width, Height, Data) -> boolean()

Types:

This = wxImage()
Width = integer()

Height = integer()
Data = binary()

See **external documentation**.
Also:
create(This, Width, Height, [Option]) -> boolean() when
This::wxImage(), Width::integer(), Height::integer(),
Option :: {clear, boolean()}.

create(This, Width, Height, Data, Alpha) -> boolean()

Types:

This = wxImage()
Width = integer()
Height = integer()
Data = binary()
Alpha = binary()

See **external documentation**.
Also:
create(This, Width, Height, Data, [Option]) -> boolean() when
This::wxImage(), Width::integer(), Height::integer(), Data::binary(),
Option :: {static_data, boolean()}.

create(This, Width, Height, Data, Alpha, Option::[Option]) -> boolean()

Types:

This = wxImage()
Width = integer()
Height = integer()
Data = binary()
Alpha = binary()
Option = {static_data, boolean()}

See **external documentation**.

Destroy(This) -> ok

Types:

This = wxImage()

See **external documentation**.

findFirstUnusedColour(This) -> Result

Types:

Result = {Res::boolean(), R::integer(), G::integer(), B::integer()}
This = wxImage()

Equivalent to *findFirstUnusedColour(This, [])*.

findFirstUnusedColour(This, Option::[Option]) -> Result

Types:

```
Result = {Res::boolean(), R::integer(), G::integer(), B::integer()}
This = wxImage()
Option = {startR, integer()} | {startG, integer()} | {startB, integer()}
```

See **external documentation**.

getImageExtWildcard() -> charlist() (see module unicode)

See **external documentation**.

getAlpha(This) -> binary()

Types:

```
This = wxImage()
```

See **external documentation**.

getAlpha(This, X, Y) -> integer()

Types:

```
This = wxImage()
X = integer()
Y = integer()
```

See **external documentation**.

getBlue(This, X, Y) -> integer()

Types:

```
This = wxImage()
X = integer()
Y = integer()
```

See **external documentation**.

getData(This) -> binary()

Types:

```
This = wxImage()
```

See **external documentation**.

getGreen(This, X, Y) -> integer()

Types:

```
This = wxImage()
X = integer()
Y = integer()
```

See **external documentation**.

getImageCount(Name) -> integer()

Types:

```
Name = chardata() (see module unicode)
```

Equivalent to *getImageCount(Name, [])*.

getImageCount(Name, Option::[Option]) -> integer()

Types:

Name = chardata() (see module unicode)

Option = {type, integer()}

See **external documentation**.

getHeight(This) -> integer()

Types:

This = wxImage()

See **external documentation**.

getMaskBlue(This) -> integer()

Types:

This = wxImage()

See **external documentation**.

getMaskGreen(This) -> integer()

Types:

This = wxImage()

See **external documentation**.

getMaskRed(This) -> integer()

Types:

This = wxImage()

See **external documentation**.

getOrFindMaskColour(This) -> Result

Types:

Result = {Res::boolean(), R::integer(), G::integer(), B::integer()}

This = wxImage()

See **external documentation**.

getPalette(This) -> wxPalette() (see module wxPalette)

Types:

This = wxImage()

See **external documentation**.

getRed(This, X, Y) -> integer()

Types:

This = wxImage()

X = integer()

Y = integer()

See **external documentation**.

getSubImage(This, Rect) -> wxImage()

Types:

This = wxImage()

Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}

See **external documentation**.

getWidth(This) -> integer()

Types:

This = wxImage()

See **external documentation**.

hasAlpha(This) -> boolean()

Types:

This = wxImage()

See **external documentation**.

hasMask(This) -> boolean()

Types:

This = wxImage()

See **external documentation**.

getOption(This, Name) -> charlist() (see module unicode)

Types:

This = wxImage()

Name = chardata() (see module unicode)

See **external documentation**.

getOptionInt(This, Name) -> integer()

Types:

This = wxImage()

Name = chardata() (see module unicode)

See **external documentation**.

hasOption(This, Name) -> boolean()

Types:

This = wxImage()

Name = chardata() (see module unicode)

See **external documentation**.

initAlpha(This) -> ok

Types:

This = wxImage()

See **external documentation**.

initStandardHandlers() -> ok

See **external documentation**.

isTransparent(This, X, Y) -> boolean()

Types:

```
This = wxImage()
X = integer()
Y = integer()
```

Equivalent to *isTransparent(This, X, Y, [])*.

isTransparent(This, X, Y, Option::[Option]) -> boolean()

Types:

```
This = wxImage()
X = integer()
Y = integer()
Option = {threshold, integer()}
```

See **external documentation**.

loadFile(This, Name) -> boolean()

Types:

```
This = wxImage()
Name = chardata() (see module unicode)
```

Equivalent to *loadFile(This, Name, [])*.

loadFile(This, Name, Option::[Option]) -> boolean()

Types:

```
This = wxImage()
Name = chardata() (see module unicode)
Option = {type, integer()} | {index, integer()}
```

See **external documentation**.

loadFile(This, Name, Mimetype, Option::[Option]) -> boolean()

Types:

```
This = wxImage()
Name = chardata() (see module unicode)
Mimetype = chardata() (see module unicode)
Option = {index, integer()}
```

See **external documentation**.

ok(This) -> boolean()

Types:

```
This = wxImage()
```

See **external documentation**.

removeHandler(Name) -> boolean()

Types:

Name = chardata() (see module unicode)

See **external documentation**.

mirror(This) -> wxImage()

Types:

This = wxImage()

Equivalent to *mirror(This, [])*.

mirror(This, Option::[Option]) -> wxImage()

Types:

This = wxImage()

Option = {horizontally, boolean()}

See **external documentation**.

replace(This, R1, G1, B1, R2, G2, B2) -> ok

Types:

This = wxImage()

R1 = integer()

G1 = integer()

B1 = integer()

R2 = integer()

G2 = integer()

B2 = integer()

See **external documentation**.

rescale(This, Width, Height) -> wxImage()

Types:

This = wxImage()

Width = integer()

Height = integer()

Equivalent to *rescale(This, Width, Height, [])*.

rescale(This, Width, Height, Option::[Option]) -> wxImage()

Types:

This = wxImage()

Width = integer()

Height = integer()

Option = {quality, integer()}

See **external documentation**.

```
resize(This, Size, Pos) -> wxImage()
```

Types:

```
This = wxImage()
Size = {W::integer(), H::integer()}
Pos = {X::integer(), Y::integer()}
```

Equivalent to *resize(This, Size, Pos, [])*.

```
resize(This, Size, Pos, Option::[Option]) -> wxImage()
```

Types:

```
This = wxImage()
Size = {W::integer(), H::integer()}
Pos = {X::integer(), Y::integer()}
Option = {r, integer()} | {g, integer()} | {b, integer()}
```

See **external documentation**.

```
rotate(This, Angle, Centre_of_rotation) -> wxImage()
```

Types:

```
This = wxImage()
Angle = number()
Centre_of_rotation = {X::integer(), Y::integer()}
```

Equivalent to *rotate(This, Angle, Centre_of_rotation, [])*.

```
rotate(This, Angle, Centre_of_rotation, Option::[Option]) -> wxImage()
```

Types:

```
This = wxImage()
Angle = number()
Centre_of_rotation = {X::integer(), Y::integer()}
Option = {interpolating, boolean()} | {offset_after_rotation,
{X::integer(), Y::integer()}}
```

See **external documentation**.

```
rotateHue(This, Angle) -> ok
```

Types:

```
This = wxImage()
Angle = number()
```

See **external documentation**.

```
rotate90(This) -> wxImage()
```

Types:

```
This = wxImage()
```

Equivalent to *rotate90(This, [])*.

rotate90(This, Option::[Option]) -> wxImage()

Types:

This = wxImage()
Option = {clockwise, boolean()}

See **external documentation**.

saveFile(This, Name) -> boolean()

Types:

This = wxImage()
Name = chardata() (see module unicode)

See **external documentation**.

saveFile(This, Name, Type) -> boolean()

Types:

This = wxImage()
Name = chardata() (see module unicode)
Type = integer()

See **external documentation**.
Also:
saveFile(This, Name, Mimetype) -> boolean() when
This::wxImage(), Name::unicode:chardata(), Mimetype::unicode:chardata().

scale(This, Width, Height) -> wxImage()

Types:

This = wxImage()
Width = integer()
Height = integer()

Equivalent to *scale(This, Width, Height, [])*.

scale(This, Width, Height, Option::[Option]) -> wxImage()

Types:

This = wxImage()
Width = integer()
Height = integer()
Option = {quality, integer()}

See **external documentation**.

size(This, Size, Pos) -> wxImage()

Types:

This = wxImage()
Size = {W::integer(), H::integer()}
Pos = {X::integer(), Y::integer()}

Equivalent to *size(This, Size, Pos, [])*.

size(This, Size, Pos, Option::[Option]) -> wxImage()

Types:

```
This = wxImage()
Size = {W::integer(), H::integer()}
Pos = {X::integer(), Y::integer()}
Option = {r, integer()} | {g, integer()} | {b, integer()}
```

See **external documentation**.

setAlpha(This, Alpha) -> ok

Types:

```
This = wxImage()
Alpha = binary()
```

Equivalent to *setAlpha(This, Alpha, [])*.

setAlpha(This, Alpha, Option::[Option]) -> ok

Types:

```
This = wxImage()
Alpha = binary()
Option = {static_data, boolean()}
```

See **external documentation**.

setAlpha(This, X, Y, Alpha) -> ok

Types:

```
This = wxImage()
X = integer()
Y = integer()
Alpha = integer()
```

See **external documentation**.

setData(This, Data) -> ok

Types:

```
This = wxImage()
Data = binary()
```

Equivalent to *setData(This, Data, [])*.

setData(This, Data, Option::[Option]) -> ok

Types:

```
This = wxImage()
Data = binary()
Option = {static_data, boolean()}
```

See **external documentation**.

setData(This, Data, New_width, New_height) -> ok

Types:

This = wxImage()
Data = binary()
New_width = integer()
New_height = integer()

Equivalent to *setData(This, Data, New_width, New_height, [])*.

setData(This, Data, New_width, New_height, Option::[Option]) -> ok

Types:

This = wxImage()
Data = binary()
New_width = integer()
New_height = integer()
Option = {static_data, boolean()}

See **external documentation**.

setMask(This) -> ok

Types:

This = wxImage()

Equivalent to *setMask(This, [])*.

setMask(This, Option::[Option]) -> ok

Types:

This = wxImage()
Option = {mask, boolean()}

See **external documentation**.

setMaskColour(This, R, G, B) -> ok

Types:

This = wxImage()
R = integer()
G = integer()
B = integer()

See **external documentation**.

setMaskFromImage(This, Mask, Mr, Mg, Mb) -> boolean()

Types:

This = wxImage()
Mask = wxImage()
Mr = integer()
Mg = integer()
Mb = integer()

See **external documentation**.

setOption(This, Name, Value) -> ok

Types:

```
This = wxImage()
Name = chardata() (see module unicode)
Value = integer()
```

See **external documentation**.
Also:
setOption(This, Name, Value) -> ok when
This::wxImage(), Name::unicode:chardata(), Value::unicode:chardata().

setPalette(This, Palette) -> ok

Types:

```
This = wxImage()
Palette = wxPalette() (see module wxPalette)
```

See **external documentation**.

setRGB(This, Rect, R, G, B) -> ok

Types:

```
This = wxImage()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
R = integer()
G = integer()
B = integer()
```

See **external documentation**.

setRGB(This, X, Y, R, G, B) -> ok

Types:

```
This = wxImage()
X = integer()
Y = integer()
R = integer()
G = integer()
B = integer()
```

See **external documentation**.

destroy(This::wxImage()) -> ok

Destroys this object, do not use object again

wxImageList

Erlang module

See external documentation: **wxImageList**.

DATA TYPES

`wxImageList()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxImageList()`

See **external documentation**.

`new(Width, Height) -> wxImageList()`

Types:

`Width = integer()`
`Height = integer()`

Equivalent to *new(Width, Height, [])*.

`new(Width, Height, Option::[Option]) -> wxImageList()`

Types:

`Width = integer()`
`Height = integer()`
`Option = {mask, boolean()} | {initialCount, integer()}`

See **external documentation**.

`add(This, Bitmap) -> integer()`

Types:

`This = wxImageList()`
`Bitmap = wxBitmap() (see module wxBitmap)`

See **external documentation**.

`add(This, Bitmap, Mask) -> integer()`

Types:

`This = wxImageList()`
`Bitmap = wxBitmap() (see module wxBitmap)`
`Mask = wxBitmap() (see module wxBitmap)`

See **external documentation**.
Also:
add(This, Bitmap, MaskColour) -> integer() when

This::wxImageList(), Bitmap::wxBitmap:wxBitmap(), MaskColour::wx:wx_colour().

create(This, Width, Height) -> boolean()

Types:

```
This = wxImageList()
Width = integer()
Height = integer()
```

Equivalent to *create(This, Width, Height, [])*.

create(This, Width, Height, Option::[Option]) -> boolean()

Types:

```
This = wxImageList()
Width = integer()
Height = integer()
Option = {mask, boolean()} | {initialCount, integer()}
```

See **external documentation**.

draw(This, Index, Dc, X, Y) -> boolean()

Types:

```
This = wxImageList()
Index = integer()
Dc = wxDC() (see module wxDC)
X = integer()
Y = integer()
```

Equivalent to *draw(This, Index, Dc, X, Y, [])*.

draw(This, Index, Dc, X, Y, Option::[Option]) -> boolean()

Types:

```
This = wxImageList()
Index = integer()
Dc = wxDC() (see module wxDC)
X = integer()
Y = integer()
Option = {flags, integer()} | {solidBackground, boolean()}
```

See **external documentation**.

getBitmap(This, Index) -> wxBitmap() (see module wxBitmap)

Types:

```
This = wxImageList()
Index = integer()
```

See **external documentation**.

getIcon(This, Index) -> wxIcon() (see module wxIcon)

Types:

This = wxImageList()
Index = integer()

See **external documentation**.

getImageCount(This) -> integer()

Types:

This = wxImageList()

See **external documentation**.

getSize(This, Index) -> Result

Types:

Result = {Res::boolean(), Width::integer(), Height::integer()}
This = wxImageList()
Index = integer()

See **external documentation**.

remove(This, Index) -> boolean()

Types:

This = wxImageList()
Index = integer()

See **external documentation**.

removeAll(This) -> boolean()

Types:

This = wxImageList()

See **external documentation**.

replace(This, Index, Bitmap) -> boolean()

Types:

This = wxImageList()
Index = integer()
Bitmap = wxBitmap() (see module wxBitmap)

See **external documentation**.

replace(This, Index, Bitmap, Mask) -> boolean()

Types:

This = wxImageList()
Index = integer()
Bitmap = wxBitmap() (see module wxBitmap)
Mask = wxBitmap() (see module wxBitmap)

See **external documentation**.

destroy(This::wxImageList()) -> ok

Destroys this object, do not use object again

wxJoystickEvent

Erlang module

See external documentation: **wxJoystickEvent**.

Use *wxEvtHandler:connect/3* with EventType:

joy_button_down, *joy_button_up*, *joy_move*, *joy_zmove*

See also the message variant *#wxJoystick{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxJoystickEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`buttonDown(This) -> boolean()`

Types:

`This = wxJoystickEvent()`

Equivalent to *buttonDown(This, [])*.

`buttonDown(This, Option::[Option]) -> boolean()`

Types:

`This = wxJoystickEvent()`
`Option = {but, integer()}`

See **external documentation**.

`buttonIsDown(This) -> boolean()`

Types:

`This = wxJoystickEvent()`

Equivalent to *buttonIsDown(This, [])*.

`buttonIsDown(This, Option::[Option]) -> boolean()`

Types:

`This = wxJoystickEvent()`
`Option = {but, integer()}`

See **external documentation**.

`buttonUp(This) -> boolean()`

Types:

`This = wxJoystickEvent()`

Equivalent to *buttonUp(This, [])*.

buttonUp(This, Option::[Option]) -> boolean()

Types:

This = wxJoystickEvent()

Option = {but, integer()}

See **external documentation**.

getButtonChange(This) -> integer()

Types:

This = wxJoystickEvent()

See **external documentation**.

getButtonState(This) -> integer()

Types:

This = wxJoystickEvent()

See **external documentation**.

getJoystick(This) -> integer()

Types:

This = wxJoystickEvent()

See **external documentation**.

getPosition(This) -> {X::integer(), Y::integer()}

Types:

This = wxJoystickEvent()

See **external documentation**.

getZPosition(This) -> integer()

Types:

This = wxJoystickEvent()

See **external documentation**.

isButton(This) -> boolean()

Types:

This = wxJoystickEvent()

See **external documentation**.

isMove(This) -> boolean()

Types:

This = wxJoystickEvent()

See **external documentation**.

isZMove(This) -> boolean()

Types:

This = wxJoystickEvent()

See **external documentation**.

wxKeyEvent

Erlang module

See external documentation: **wxKeyEvent**.

Use *wxEvtHandler:connect/3* with EventType:

char, *char_hook*, *key_down*, *key_up*

See also the message variant *#wxKey{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxKeyEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`altDown(This) -> boolean()`

Types:

`This = wxKeyEvent()`

See **external documentation**.

`cmdDown(This) -> boolean()`

Types:

`This = wxKeyEvent()`

See **external documentation**.

`controlDown(This) -> boolean()`

Types:

`This = wxKeyEvent()`

See **external documentation**.

`getKeyCode(This) -> integer()`

Types:

`This = wxKeyEvent()`

See **external documentation**.

`getModifiers(This) -> integer()`

Types:

`This = wxKeyEvent()`

See **external documentation**.

getPosition(This) -> {X::integer(), Y::integer()}

Types:

This = wxKeyEvent()

See **external documentation**.

getRawKeyCode(This) -> integer()

Types:

This = wxKeyEvent()

See **external documentation**.

getRawKeyFlags(This) -> integer()

Types:

This = wxKeyEvent()

See **external documentation**.

getUnicodeKey(This) -> integer()

Types:

This = wxKeyEvent()

See **external documentation**.

getX(This) -> integer()

Types:

This = wxKeyEvent()

See **external documentation**.

getY(This) -> integer()

Types:

This = wxKeyEvent()

See **external documentation**.

hasModifiers(This) -> boolean()

Types:

This = wxKeyEvent()

See **external documentation**.

metaDown(This) -> boolean()

Types:

This = wxKeyEvent()

See **external documentation**.

shiftDown(This) -> boolean()

Types:

This = wxKeyEvent()

See **external documentation**.

wxLayoutAlgorithm

Erlang module

See external documentation: **wxLayoutAlgorithm**.

DATA TYPES

`wxLayoutAlgorithm()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxLayoutAlgorithm()`

See **external documentation**.

`layoutFrame(This, Frame) -> boolean()`

Types:

```
This = wxLayoutAlgorithm()
Frame = wxFrame() (see module wxFrame)
```

Equivalent to *layoutFrame(This, Frame, [])*.

`layoutFrame(This, Frame, Option::[Option]) -> boolean()`

Types:

```
This = wxLayoutAlgorithm()
Frame = wxFrame() (see module wxFrame)
Option = {mainWindow, wxWindow() (see module wxWindow)}
```

See **external documentation**.

`layoutMDIFrame(This, Frame) -> boolean()`

Types:

```
This = wxLayoutAlgorithm()
Frame = wxMDIParentFrame() (see module wxMDIParentFrame)
```

Equivalent to *layoutMDIFrame(This, Frame, [])*.

`layoutMDIFrame(This, Frame, Option::[Option]) -> boolean()`

Types:

```
This = wxLayoutAlgorithm()
Frame = wxMDIParentFrame() (see module wxMDIParentFrame)
Option = {rect, {X::integer(), Y::integer(), W::integer(), H::integer()}}
```

See **external documentation**.

layoutWindow(This, Frame) -> boolean()

Types:

```
This = wxLayoutAlgorithm()
Frame = wxWindow() (see module wxWindow)
```

Equivalent to *layoutWindow(This, Frame, [])*.

layoutWindow(This, Frame, Option::[Option]) -> boolean()

Types:

```
This = wxLayoutAlgorithm()
Frame = wxWindow() (see module wxWindow)
Option = {mainWindow, wxWindow() (see module wxWindow)}
```

See **external documentation**.

destroy(This::wxLayoutAlgorithm()) -> ok

Destroys this object, do not use object again

wxListBox

Erlang module

See external documentation: **wxListBox**.

This class is derived (and can use functions) from:
wxControlWithItems
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxListBox()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxListBox()`

See **external documentation**.

`new(Parent, Id) -> wxListBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxListBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {choices, [chardata() (see module unicode)]} | {style,
integer()} | {validator, wx_object() (see module wx)}
```

See **external documentation**.

`create(This, Parent, Id, Pos, Size, Choices) -> boolean()`

Types:

```
This = wxListBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
```

Equivalent to *create(This, Parent, Id, Pos, Size, Choices, [])*.

create(This, Parent, Id, Pos, Size, Choices, Option::[Option]) -> boolean()

Types:

```
This = wxListBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
Option = {style, integer()} | {validator, wx_object() (see module wx)}
```

See **external documentation**.

deselect(This, N) -> ok

Types:

```
This = wxListBox()
N = integer()
```

See **external documentation**.

getSelections(This) -> Result

Types:

```
Result = {Res::integer(), ASelections::[integer()]}
This = wxListBox()
```

See **external documentation**.

insertItems(This, Items, Pos) -> ok

Types:

```
This = wxListBox()
Items = [chardata() (see module unicode)]
Pos = integer()
```

See **external documentation**.

isSelected(This, N) -> boolean()

Types:

```
This = wxListBox()
N = integer()
```

See **external documentation**.

set(This, Items) -> ok

Types:

```
This = wxListBox()
Items = [chardata() (see module unicode)]
```

See **external documentation**.

hitTest(This, Point) -> integer()

Types:

This = wxListBox()

Point = {X::integer(), Y::integer()}

See **external documentation**.

setFirstItem(This, N) -> ok

Types:

This = wxListBox()

N = integer()

See **external documentation**.
Also:
setFirstItem(This, S) -> ok when
This::wxListBox(), S::unicode:chardata().

destroy(This::wxListBox()) -> ok

Destroys this object, do not use object again

wxListCtrl

Erlang module

See external documentation: **wxListCtrl**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxListCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxListCtrl()`

See **external documentation**.

`new(Parent::wxWindow() (see module wxWindow)) -> wxListCtrl()`

Equivalent to *new(Parent, [])*.

`new(Parent::wxWindow() (see module wxWindow), Options::[Option]) -> wxListCtrl()`

Types:

```
Option = {winid, integer()} | {pos, {X::integer(), Y::integer()}} |
{size, {W::integer(), H::integer()}} | {style, integer()} | {validator,
wx() (see module wx)} | {onGetItemText, OnGetItemText} | {onGetItemAttr,
OnGetItemAttr} | {onGetItemColumnImage, OnGetItemColumnImage}
OnGetItemText = (This, Item, Column) -> wxString()
OnGetItemAttr = (This, Item) -> wxListItemAttr()
OnGetItemColumnImage = (This, Item, Column) -> integer()
```

See **external documentation**.

`arrange(This) -> boolean()`

Types:

```
This = wxListCtrl()
```

Equivalent to *arrange(This, [])*.

`arrange(This, Option::[Option]) -> boolean()`

Types:

```
This = wxListCtrl()
Option = {flag, integer()}
```

See **external documentation**.

assignImageList(This, ImageList, Which) -> ok

Types:

This = wxListCtrl()

ImageList = wxImageList() (see module wxImageList)

Which = integer()

See **external documentation**.

clearAll(This) -> ok

Types:

This = wxListCtrl()

See **external documentation**.

create(This::wxListCtrl(), Parent::wxWindow() (see module wxWindow)) -> bool()

Equivalent to *create(This, Parent, [])*.

create(This::wxListCtrl(), Parent::wxWindow() (see module wxWindow), Options::[Option]) -> bool()

Types:

Option = {winid, integer()} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {validator, wx() (see module wx)} | {onGetItemText, OnGetItemText} | {onGetItemAttr, OnGetItemAttr} | {onGetItemColumnImage, OnGetItemColumnImage}

OnGetItemText = (This, Item, Column) -> wxString()

OnGetItemAttr = (This, Item) -> wxListItemAttr()

OnGetItemColumnImage = (This, Item, Column) -> integer()

See **external documentation**.

deleteAllItems(This) -> boolean()

Types:

This = wxListCtrl()

See **external documentation**.

deleteColumn(This, Col) -> boolean()

Types:

This = wxListCtrl()

Col = integer()

See **external documentation**.

deleteItem(This, Item) -> boolean()

Types:

This = wxListCtrl()

```
Item = integer()
```

See **external documentation**.

editLabel(This, Item) -> wxTextCtrl() (see module wxTextCtrl)

Types:

```
This = wxListCtrl()
Item = integer()
```

See **external documentation**.

ensureVisible(This, Item) -> boolean()

Types:

```
This = wxListCtrl()
Item = integer()
```

See **external documentation**.

findItem(This, Start, Str) -> integer()

Types:

```
This = wxListCtrl()
Start = integer()
Str = chardata() (see module unicode)
```

Equivalent to *findItem(This, Start, Str, [])*.

findItem(This, Start, Str, Option::[Option]) -> integer()

Types:

```
This = wxListCtrl()
Start = integer()
Str = chardata() (see module unicode)
Option = {partial, boolean()}
```

See **external documentation**.
Also:
findItem(This, Start, Pt, Direction) -> integer() when
This::wxListCtrl(), Start::integer(), Pt::{X::integer(), Y::integer()}, Direction::integer().

getColumn(This, Col, Item) -> boolean()

Types:

```
This = wxListCtrl()
Col = integer()
Item = wxListItem() (see module wxListItem)
```

See **external documentation**.

getColumnCount(This) -> integer()

Types:

```
This = wxListCtrl()
```

See **external documentation**.

getColumnWidth(This, Col) -> integer()

Types:

This = wxListCtrl()

Col = integer()

See **external documentation**.

getCountPerPage(This) -> integer()

Types:

This = wxListCtrl()

See **external documentation**.

getEditControl(This) -> wxTextCtrl() (see module wxTextCtrl)

Types:

This = wxListCtrl()

See **external documentation**.

getImageList(This, Which) -> wxImageList() (see module wxImageList)

Types:

This = wxListCtrl()

Which = integer()

See **external documentation**.

getItem(This, Info) -> boolean()

Types:

This = wxListCtrl()

Info = wxListItem() (see module wxListItem)

See **external documentation**.

getItemBackgroundColour(This, Item) -> wx_colour4() (see module wx)

Types:

This = wxListCtrl()

Item = integer()

See **external documentation**.

getItemCount(This) -> integer()

Types:

This = wxListCtrl()

See **external documentation**.

getItemData(This, Item) -> integer()

Types:

```
This = wxListCtrl()
Item = integer()
```

See **external documentation**.

getItemFont(This, Item) -> wxFont() (see module wxFont)

Types:

```
This = wxListCtrl()
Item = integer()
```

See **external documentation**.

getItemPosition(This, Item) -> Result

Types:

```
Result = {Res::boolean(), Pos::{X::integer(), Y::integer()}}
This = wxListCtrl()
Item = integer()
```

See **external documentation**.

getItemRect(This, Item) -> Result

Types:

```
Result = {Res::boolean(), Rect::{X::integer(), Y::integer(), W::integer(),
H::integer()}}
This = wxListCtrl()
Item = integer()
```

Equivalent to *getItemRect(This, Item, [])*.

getItemRect(This, Item, Option::[Option]) -> Result

Types:

```
Result = {Res::boolean(), Rect::{X::integer(), Y::integer(), W::integer(),
H::integer()}}
This = wxListCtrl()
Item = integer()
Option = {code, integer()}
```

See **external documentation**.

getItemSpacing(This) -> {W::integer(), H::integer()}

Types:

```
This = wxListCtrl()
```

See **external documentation**.

getItemState(This, Item, StateMask) -> integer()

Types:

```
This = wxListCtrl()
Item = integer()
```

```
StateMask = integer()
```

See **external documentation**.

getItemText(This, Item) -> charlist() (see module unicode)

Types:

```
This = wxListCtrl()
Item = integer()
```

See **external documentation**.

getItemTextColour(This, Item) -> wx_colour4() (see module wx)

Types:

```
This = wxListCtrl()
Item = integer()
```

See **external documentation**.

getNextItem(This, Item) -> integer()

Types:

```
This = wxListCtrl()
Item = integer()
```

Equivalent to *getNextItem(This, Item, [])*.

getNextItem(This, Item, Option::[Option]) -> integer()

Types:

```
This = wxListCtrl()
Item = integer()
Option = {geometry, integer()} | {state, integer()}
```

See **external documentation**.

getSelectedItemCount(This) -> integer()

Types:

```
This = wxListCtrl()
```

See **external documentation**.

getTextColour(This) -> wx_colour4() (see module wx)

Types:

```
This = wxListCtrl()
```

See **external documentation**.

getTopItem(This) -> integer()

Types:

```
This = wxListCtrl()
```

See **external documentation**.

getViewRect(This) -> {X::integer(), Y::integer(), W::integer(), H::integer()}

Types:

This = wxListCtrl()

See **external documentation**.

hitTest(This, Point, Flags) -> integer()

Types:

This = wxListCtrl()
Point = {X::integer(), Y::integer()}
Flags = integer()

See **external documentation**.

insertColumn(This, Col, Heading) -> integer()

Types:

This = wxListCtrl()
Col = integer()
Heading = chardata() (see module unicode)

See **external documentation**.
Also:
insertColumn(This, Col, Info) -> integer() when
This::wxListCtrl(), Col::integer(), Info::wxListItem:wxListItem().

insertColumn(This, Col, Heading, Option::[Option]) -> integer()

Types:

This = wxListCtrl()
Col = integer()
Heading = chardata() (see module unicode)
Option = {format, integer()} | {width, integer()}

See **external documentation**.

insertItem(This, Info) -> integer()

Types:

This = wxListCtrl()
Info = wxListItem() (see module wxListItem)

See **external documentation**.

insertItem(This, Index, ImageIndex) -> integer()

Types:

This = wxListCtrl()
Index = integer()
ImageIndex = integer()

See **external documentation**.
Also:
insertItem(This, Index, Label) -> integer() when

This::wxListCtrl(), Index::integer(), Label::unicode:chardata().

insertItem(This, Index, Label, ImageIndex) -> integer()

Types:

```
This = wxListCtrl()
Index = integer()
Label = chardata() (see module unicode)
ImageIndex = integer()
```

See **external documentation**.

refreshItem(This, Item) -> ok

Types:

```
This = wxListCtrl()
Item = integer()
```

See **external documentation**.

refreshItems(This, ItemFrom, ItemTo) -> ok

Types:

```
This = wxListCtrl()
ItemFrom = integer()
ItemTo = integer()
```

See **external documentation**.

scrollList(This, Dx, Dy) -> boolean()

Types:

```
This = wxListCtrl()
Dx = integer()
Dy = integer()
```

See **external documentation**.

setBackgroundColour(This, Colour) -> boolean()

Types:

```
This = wxListCtrl()
Colour = wx_colour() (see module wx)
```

See **external documentation**.

setColumn(This, Col, Item) -> boolean()

Types:

```
This = wxListCtrl()
Col = integer()
Item = wxListItem() (see module wxListItem)
```

See **external documentation**.

setColumnWidth(This, Col, Width) -> boolean()

Types:

```
This = wxListCtrl()
Col = integer()
Width = integer()
```

See **external documentation**.

setImageList(This, ImageList, Which) -> ok

Types:

```
This = wxListCtrl()
ImageList = wxImageList() (see module wxImageList)
Which = integer()
```

See **external documentation**.

setItem(This, Info) -> boolean()

Types:

```
This = wxListCtrl()
Info = wxListItem() (see module wxListItem)
```

See **external documentation**.

setItem(This, Index, Col, Label) -> integer()

Types:

```
This = wxListCtrl()
Index = integer()
Col = integer()
Label = chardata() (see module unicode)
```

Equivalent to *setItem(This, Index, Col, Label, [])*.

setItem(This, Index, Col, Label, Option::[Option]) -> integer()

Types:

```
This = wxListCtrl()
Index = integer()
Col = integer()
Label = chardata() (see module unicode)
Option = {imageId, integer()}
```

See **external documentation**.

setItemBackgroundColour(This, Item, Col) -> ok

Types:

```
This = wxListCtrl()
Item = integer()
Col = wx_colour() (see module wx)
```

See **external documentation**.

setItemCount(This, Count) -> ok

Types:

This = wxListCtrl()
Count = integer()

See **external documentation**.

setItemData(This, Item, Data) -> boolean()

Types:

This = wxListCtrl()
Item = integer()
Data = integer()

See **external documentation**.

setItemFont(This, Item, F) -> ok

Types:

This = wxListCtrl()
Item = integer()
F = wxFont() (see module wxFont)

See **external documentation**.

setItemImage(This, Item, Image) -> boolean()

Types:

This = wxListCtrl()
Item = integer()
Image = integer()

Equivalent to *setItemImage(This, Item, Image, [])*.

setItemImage(This, Item, Image, Option::[Option]) -> boolean()

Types:

This = wxListCtrl()
Item = integer()
Image = integer()
Option = {selImage, integer()}

See **external documentation**.

setItemColumnImage(This, Item, Column, Image) -> boolean()

Types:

This = wxListCtrl()
Item = integer()
Column = integer()
Image = integer()

See **external documentation**.

setItemPosition(This, Item, Pos) -> boolean()

Types:

```
This = wxListCtrl()
Item = integer()
Pos = {X::integer(), Y::integer()}
```

See **external documentation**.

setItemState(This, Item, State, StateMask) -> boolean()

Types:

```
This = wxListCtrl()
Item = integer()
State = integer()
StateMask = integer()
```

See **external documentation**.

setItemText(This, Item, Str) -> ok

Types:

```
This = wxListCtrl()
Item = integer()
Str = chardata() (see module unicode)
```

See **external documentation**.

setItemTextColour(This, Item, Col) -> ok

Types:

```
This = wxListCtrl()
Item = integer()
Col = wx_colour() (see module wx)
```

See **external documentation**.

setSingleStyle(This, Style) -> ok

Types:

```
This = wxListCtrl()
Style = integer()
```

Equivalent to *setSingleStyle(This, Style, [])*.

setSingleStyle(This, Style, Option::[Option]) -> ok

Types:

```
This = wxListCtrl()
Style = integer()
Option = {add, boolean()}
```

See **external documentation**.

setTextColour(This, Col) -> ok

Types:

This = wxListCtrl()
Col = wx_colour() (see module wx)

See **external documentation**.

setWindowStyleFlag(This, Style) -> ok

Types:

This = wxListCtrl()
Style = integer()

See **external documentation**.

sortItems(This::wxListCtrl(), SortCallBack::function()) -> boolean()

Sort the items in the list control

```
SortCallBack(Item1,Item2) -> integer()
```

SortCallBack receives the client data associated with two items to compare, and should return 0 if the items are equal, a negative value if the first item is less than the second one and a positive value if the first item is greater than the second one.
NOTE: The callback may not call other (wx) processes.

destroy(This::wxListCtrl()) -> ok

Destroys this object, do not use object again

wxListEvent

Erlang module

See external documentation: **wxListEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_list_begin_drag, *command_list_begin_rdrag*, *command_list_begin_label_edit*, *command_list_end_label_edit*, *command_list_delete_item*, *command_list_delete_all_items*, *command_list_key_down*, *command_list_insert_item*, *command_list_col_click*, *command_list_col_right_click*, *command_list_col_begin_drag*, *command_list_col_dragging*, *command_list_col_end_drag*, *command_list_item_selected*, *command_list_item_deselected*, *command_list_item_right_click*, *command_list_item_middle_click*, *command_list_item_activated*, *command_list_item_focused*, *command_list_cache_hint*

See also the message variant *#wxList{}* event record type.

This class is derived (and can use functions) from:
wxNotifyEvent
wxCommandEvent
wxEvent

DATA TYPES

`wxListEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getCacheFrom(This) -> integer()`

Types:

`This = wxListEvent()`

See **external documentation**.

`getCacheTo(This) -> integer()`

Types:

`This = wxListEvent()`

See **external documentation**.

`getKeyCode(This) -> integer()`

Types:

`This = wxListEvent()`

See **external documentation**.

`getIndex(This) -> integer()`

Types:

`This = wxListEvent()`

See **external documentation**.

getColumn(This) -> integer()

Types:

This = wxListEvent()

See **external documentation**.

getPoint(This) -> {X::integer(), Y::integer()}

Types:

This = wxListEvent()

See **external documentation**.

getLabel(This) -> charlist() (see module unicode)

Types:

This = wxListEvent()

See **external documentation**.

getText(This) -> charlist() (see module unicode)

Types:

This = wxListEvent()

See **external documentation**.

getImage(This) -> integer()

Types:

This = wxListEvent()

See **external documentation**.

getData(This) -> integer()

Types:

This = wxListEvent()

See **external documentation**.

getMask(This) -> integer()

Types:

This = wxListEvent()

See **external documentation**.

getItem(This) -> wxListItem() (see module wxListItem)

Types:

This = wxListEvent()

See **external documentation**.

```
isEditCancelled(This) -> boolean()
```

Types:

```
This = wxListEvent()
```

See **external documentation**.

wxListItem

Erlang module

See external documentation: **wxListItem**.

DATA TYPES

`wxListItem()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxListItem()`

See **external documentation**.

`new(Item) -> wxListItem()`

Types:

`Item = wxListItem()`

See **external documentation**.

`clear(This) -> ok`

Types:

`This = wxListItem()`

See **external documentation**.

`getAlign(This) -> wx_enum() (see module wx)`

Types:

`This = wxListItem()`

See **external documentation**.
Res = ?wxLIST_FORMAT_LEFT | ?wxLIST_FORMAT_RIGHT | ?wxLIST_FORMAT_CENTRE | ?wxLIST_FORMAT_CENTER

`getBackgroundColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxListItem()`

See **external documentation**.

`getColumn(This) -> integer()`

Types:

`This = wxListItem()`

See **external documentation**.

getFont(This) -> wxFont() (see module wxFont)

Types:

This = wxListItem()

See **external documentation**.

getId(This) -> integer()

Types:

This = wxListItem()

See **external documentation**.

getImage(This) -> integer()

Types:

This = wxListItem()

See **external documentation**.

getMask(This) -> integer()

Types:

This = wxListItem()

See **external documentation**.

getState(This) -> integer()

Types:

This = wxListItem()

See **external documentation**.

getText(This) -> charlist() (see module unicode)

Types:

This = wxListItem()

See **external documentation**.

getTextColour(This) -> wx_colour4() (see module wx)

Types:

This = wxListItem()

See **external documentation**.

getWidth(This) -> integer()

Types:

This = wxListItem()

See **external documentation**.

setAlign(This, Align) -> ok

Types:

This = wxListItem()

```
Align = wx_enum() (see module wx)
```

See **external documentation**.
Align = ?wxLIST_FORMAT_LEFT | ?wxLIST_FORMAT_RIGHT | ?wxLIST_FORMAT_CENTRE | ?wxLIST_FORMAT_CENTER

setBackgroundColour(This, ColBack) -> ok

Types:

```
This = wxListItem()
ColBack = wx_colour() (see module wx)
```

See **external documentation**.

setColumn(This, Col) -> ok

Types:

```
This = wxListItem()
Col = integer()
```

See **external documentation**.

setFont(This, Font) -> ok

Types:

```
This = wxListItem()
Font = wxFont() (see module wxFont)
```

See **external documentation**.

setId(This, Id) -> ok

Types:

```
This = wxListItem()
Id = integer()
```

See **external documentation**.

setImage(This, Image) -> ok

Types:

```
This = wxListItem()
Image = integer()
```

See **external documentation**.

setMask(This, Mask) -> ok

Types:

```
This = wxListItem()
Mask = integer()
```

See **external documentation**.

setState(This, State) -> ok

Types:

```
This = wxListItem()
State = integer()
```

See **external documentation**.

setStateMask(This, StateMask) -> ok

Types:

```
This = wxListItem()
StateMask = integer()
```

See **external documentation**.

setText(This, Text) -> ok

Types:

```
This = wxListItem()
Text = chardata() (see module unicode)
```

See **external documentation**.

setTextColour(This, ColText) -> ok

Types:

```
This = wxListItem()
ColText = wx_colour() (see module wx)
```

See **external documentation**.

setWidth(This, Width) -> ok

Types:

```
This = wxListItem()
Width = integer()
```

See **external documentation**.

destroy(This::wxListItem()) -> ok

Destroys this object, do not use object again

wxListItemAttr

Erlang module

See external documentation: **wxListItemAttr**.

DATA TYPES

`wxListItemAttr()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxListItemAttr()`

See **external documentation**.

`new(ColText, ColBack, Font) -> wxListItemAttr()`

Types:

`ColText = wx_colour() (see module wx)`
`ColBack = wx_colour() (see module wx)`
`Font = wxFont() (see module wxFont)`

See **external documentation**.

`getBackgroundColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxListItemAttr()`

See **external documentation**.

`getFont(This) -> wxFont() (see module wxFont)`

Types:

`This = wxListItemAttr()`

See **external documentation**.

`getTextColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxListItemAttr()`

See **external documentation**.

`hasBackgroundColour(This) -> boolean()`

Types:

`This = wxListItemAttr()`

See **external documentation**.

hasFont(This) -> boolean()

Types:

This = wxListItemAttr()

See **external documentation**.

hasTextColour(This) -> boolean()

Types:

This = wxListItemAttr()

See **external documentation**.

setBackgroundColour(This, ColBack) -> ok

Types:

This = wxListItemAttr()

ColBack = wx_colour() (see module wx)

See **external documentation**.

setFont(This, Font) -> ok

Types:

This = wxListItemAttr()

Font = wxFont() (see module wxFont)

See **external documentation**.

setTextColour(This, ColText) -> ok

Types:

This = wxListItemAttr()

ColText = wx_colour() (see module wx)

See **external documentation**.

destroy(This::wxListItemAttr()) -> ok

Destroys this object, do not use object again

wxListView

Erlang module

See external documentation: **wxListView**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxListView()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`clearColumnImage(This, Col) -> ok`

Types:

`This = wxListView()`
`Col = integer()`

See **external documentation**.

`focus(This, Index) -> ok`

Types:

`This = wxListView()`
`Index = integer()`

See **external documentation**.

`getFirstSelected(This) -> integer()`

Types:

`This = wxListView()`

See **external documentation**.

`getFocusedItem(This) -> integer()`

Types:

`This = wxListView()`

See **external documentation**.

`getNextSelected(This, Item) -> integer()`

Types:

`This = wxListView()`
`Item = integer()`

See **external documentation**.

isSelected(This, Index) -> boolean()

Types:

```
This = wxListView()
Index = integer()
```

See **external documentation**.

select(This, N) -> ok

Types:

```
This = wxListView()
N = integer()
```

Equivalent to *select(This, N, [])*.

select(This, N, Option::[Option]) -> ok

Types:

```
This = wxListView()
N = integer()
Option = {on, boolean()}
```

See **external documentation**.

setColumnImage(This, Col, Image) -> ok

Types:

```
This = wxListView()
Col = integer()
Image = integer()
```

See **external documentation**.

wxListbook

Erlang module

See external documentation: **wxListbook**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxListbook()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxListbook()`

See **external documentation**.

`new(Parent, Id) -> wxListbook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxListbook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`addPage(This, Page, Text) -> boolean()`

Types:

```
This = wxListbook()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
```

Equivalent to *addPage(This, Page, Text, [])*.

`addPage(This, Page, Text, Option::[Option]) -> boolean()`

Types:

```
This = wxListbook()
```

```
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}
```

See **external documentation**.

advanceSelection(This) -> ok

Types:

```
This = wxListbook()
```

Equivalent to *advanceSelection(This, [])*.

advanceSelection(This, Option::[Option]) -> ok

Types:

```
This = wxListbook()
Option = {forward, boolean()}
```

See **external documentation**.

assignImageList(This, ImageList) -> ok

Types:

```
This = wxListbook()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

create(This, Parent, Id) -> boolean()

Types:

```
This = wxListbook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

create(This, Parent, Id, Option::[Option]) -> boolean()

Types:

```
This = wxListbook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

deleteAllPages(This) -> boolean()

Types:

```
This = wxListbook()
```

See **external documentation**.

deletePage(This, N) -> boolean()

Types:

This = wxListbook()

N = integer()

See **external documentation**.

removePage(This, N) -> boolean()

Types:

This = wxListbook()

N = integer()

See **external documentation**.

getCurrentPage(This) -> wxWindow() (see module wxWindow)

Types:

This = wxListbook()

See **external documentation**.

getImageList(This) -> wxImageList() (see module wxImageList)

Types:

This = wxListbook()

See **external documentation**.

getPage(This, N) -> wxWindow() (see module wxWindow)

Types:

This = wxListbook()

N = integer()

See **external documentation**.

getPageCount(This) -> integer()

Types:

This = wxListbook()

See **external documentation**.

getPageImage(This, N) -> integer()

Types:

This = wxListbook()

N = integer()

See **external documentation**.

getPageText(This, N) -> charlist() (see module unicode)

Types:

This = wxListbook()

N = integer()

See **external documentation**.

getSelection(This) -> integer()

Types:

This = wxListbook()

See **external documentation**.

hitTest(This, Pt) -> Result

Types:

Result = {Res::integer(), Flags::integer()}
This = wxListbook()
Pt = {X::integer(), Y::integer()}

See **external documentation**.

insertPage(This, N, Page, Text) -> boolean()

Types:

This = wxListbook()
N = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)

Equivalent to *insertPage(This, N, Page, Text, [])*.

insertPage(This, N, Page, Text, Option::[Option]) -> boolean()

Types:

This = wxListbook()
N = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}

See **external documentation**.

setImageList(This, ImageList) -> ok

Types:

This = wxListbook()
ImageList = wxImageList() (see module wxImageList)

See **external documentation**.

setPageSize(This, Size) -> ok

Types:

This = wxListbook()
Size = {W::integer(), H::integer()}

See **external documentation**.

setPageImage(This, N, ImageId) -> boolean()

Types:

This = wxListbook()
N = integer()
ImageId = integer()

See **external documentation**.

setPageText(This, N, StrText) -> boolean()

Types:

This = wxListbook()
N = integer()
StrText = chardata() (see module unicode)

See **external documentation**.

setSelection(This, N) -> integer()

Types:

This = wxListbook()
N = integer()

See **external documentation**.

changeSelection(This, N) -> integer()

Types:

This = wxListbook()
N = integer()

See **external documentation**.

destroy(This::wxListbook()) -> ok

Destroys this object, do not use object again

wxLogNull

Erlang module

See external documentation: **wxLogNull**.

DATA TYPES

`wxLogNull()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxLogNull()`

See **external documentation**.

`destroy(This::wxLogNull()) -> ok`

Destroys this object, do not use object again

wxMDIChildFrame

Erlang module

See external documentation: **wxMDIChildFrame**.

This class is derived (and can use functions) from:
wxFrame
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxMDIChildFrame()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMDIChildFrame()`

See **external documentation**.

`new(Parent, Id, Title) -> wxMDIChildFrame()`

Types:

```
Parent = wxMDIParentFrame() (see module wxMDIParentFrame)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Title, [])*.

`new(Parent, Id, Title, Option::[Option]) -> wxMDIChildFrame()`

Types:

```
Parent = wxMDIParentFrame() (see module wxMDIParentFrame)
Id = integer()
Title = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`activate(This) -> ok`

Types:

```
This = wxMDIChildFrame()
```

See **external documentation**.

`create(This, Parent, Id, Title) -> boolean()`

Types:

```
This = wxMDIChildFrame()
Parent = wxMDIParentFrame() (see module wxMDIParentFrame)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Title, [])*.

create(This, Parent, Id, Title, Option::[Option]) -> boolean()

Types:

```
This = wxMDIChildFrame()
Parent = wxMDIParentFrame() (see module wxMDIParentFrame)
Id = integer()
Title = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

maximize(This) -> ok

Types:

```
This = wxMDIChildFrame()
```

Equivalent to *maximize(This, [])*.

maximize(This, Option::[Option]) -> ok

Types:

```
This = wxMDIChildFrame()
Option = {maximize, boolean()}
```

See **external documentation**.

restore(This) -> ok

Types:

```
This = wxMDIChildFrame()
```

See **external documentation**.

destroy(This::wxMDIChildFrame()) -> ok

Destroys this object, do not use object again

wxMDIClientWindow

Erlang module

See external documentation: **wxMDIClientWindow**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxMDIClientWindow()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMDIClientWindow()`

See **external documentation**.

`new(Parent) -> wxMDIClientWindow()`

Types:

`Parent = wxMDIParentFrame() (see module wxMDIParentFrame)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxMDIClientWindow()`

Types:

`Parent = wxMDIParentFrame() (see module wxMDIParentFrame)`
`Option = {style, integer()}`

See **external documentation**.

`createClient(This, Parent) -> boolean()`

Types:

`This = wxMDIClientWindow()`
`Parent = wxMDIParentFrame() (see module wxMDIParentFrame)`

Equivalent to *createClient(This, Parent, [])*.

`createClient(This, Parent, Option::[Option]) -> boolean()`

Types:

`This = wxMDIClientWindow()`
`Parent = wxMDIParentFrame() (see module wxMDIParentFrame)`
`Option = {style, integer()}`

See **external documentation**.

destroy(This::wxMDIClientWindow()) -> ok

Destroys this object, do not use object again

wxMDIParentFrame

Erlang module

See external documentation: **wxMDIParentFrame**.

This class is derived (and can use functions) from:
wxFrame
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxMDIParentFrame()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMDIParentFrame()`

See **external documentation**.

`new(Parent, Id, Title) -> wxMDIParentFrame()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Title, [])*.

`new(Parent, Id, Title, Option::[Option]) -> wxMDIParentFrame()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`activateNext(This) -> ok`

Types:

```
This = wxMDIParentFrame()
```

See **external documentation**.

`activatePrevious(This) -> ok`

Types:

```
This = wxMDIParentFrame()
```

See **external documentation**.

arrangeIcons(This) -> ok

Types:

```
This = wxMDIParentFrame()
```

See **external documentation**.

cascade(This) -> ok

Types:

```
This = wxMDIParentFrame()
```

See **external documentation**.

create(This, Parent, Id, Title) -> boolean()

Types:

```
This = wxMDIParentFrame()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Title, [])*.

create(This, Parent, Id, Title, Option::[Option]) -> boolean()

Types:

```
This = wxMDIParentFrame()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

getActiveChild(This) -> wxMDIChildFrame() (see module wxMDIChildFrame)

Types:

```
This = wxMDIParentFrame()
```

See **external documentation**.

getClientWindow(This) -> wxMDIClientWindow() (see module wxMDIClientWindow)

Types:

```
This = wxMDIParentFrame()
```

See **external documentation**.

tile(This) -> ok

Types:

```
This = wxMDIParentFrame()
```

Equivalent to *tile(This, [])*.

tile(This, Option::[Option]) -> ok

Types:

```
This = wxMDIParentFrame()
Option = {orient, wx_enum() (see module wx)}
```

See **external documentation**.
Orient = ?wxHORIZONTAL | ?wxVERTICAL | ?wxBOTH

destroy(This::wxMDIParentFrame()) -> ok

Destroys this object, do not use object again

wxMask

Erlang module

See external documentation: **wxMask**.

DATA TYPES

`wxMask()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMask()`

See **external documentation**.

`new(Bitmap) -> wxMask()`

Types:

`Bitmap = wxBitmap() (see module wxBitmap)`

See **external documentation**.

`new(Bitmap, PaletteIndex) -> wxMask()`

Types:

`Bitmap = wxBitmap() (see module wxBitmap)`
`PaletteIndex = integer()`

See **external documentation**.
Also:
new(Bitmap, Colour) -> wxMask() when
Bitmap::wxBitmap:wxBitmap(), Colour::wx:wx_colour().

`create(This, Bitmap) -> boolean()`

Types:

`This = wxMask()`
`Bitmap = wxBitmap() (see module wxBitmap)`

See **external documentation**.

`create(This, Bitmap, PaletteIndex) -> boolean()`

Types:

`This = wxMask()`
`Bitmap = wxBitmap() (see module wxBitmap)`
`PaletteIndex = integer()`

See **external documentation**.
Also:
create(This, Bitmap, Colour) -> boolean() when

This::wxMask(), Bitmap::wxBitmap:wxBitmap(), Colour::wx:wx_colour().

`destroy(This::wxMask()) -> ok`

Destroys this object, do not use object again

wxMaximizeEvent

Erlang module

See external documentation: **wxMaximizeEvent**.

Use *wxEvtHandler:connect/3* with EventType:

maximize

See also the message variant *#wxMaximize{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxMaximizeEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxMemoryDC

Erlang module

See external documentation: **wxMemoryDC**.

This class is derived (and can use functions) from:
wxDC

DATA TYPES

`wxMemoryDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMemoryDC()`

See **external documentation**.

`new(Dc) -> wxMemoryDC()`

Types:

`Dc = wxDC() (see module wxDC) | wxBitmap() (see module wxBitmap)`

See **external documentation**.

`selectObject(This, Bmp) -> ok`

Types:

`This = wxMemoryDC()`
`Bmp = wxBitmap() (see module wxBitmap)`

See **external documentation**.

`selectObjectAsSource(This, Bmp) -> ok`

Types:

`This = wxMemoryDC()`
`Bmp = wxBitmap() (see module wxBitmap)`

See **external documentation**.

`destroy(This::wxMemoryDC()) -> ok`

Destroys this object, do not use object again

wxMenu

Erlang module

See external documentation: **wxMenu**.

This class is derived (and can use functions) from:
wxEvtHandler

DATA TYPES

`wxMenu()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMenu()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxMenu()`

Types:

`Option = {style, integer()}`

See **external documentation**.

`new(Title, Option::[Option]) -> wxMenu()`

Types:

`Title = chardata() (see module unicode)`
`Option = {style, integer()}`

See **external documentation**.

`append(This, Item) -> wxMenuItem() (see module wxMenuItem)`

Types:

`This = wxMenu()`
`Item = wxMenuItem() (see module wxMenuItem)`

See **external documentation**.

`append(This, Itemid, Text) -> wxMenuItem() (see module wxMenuItem)`

Types:

`This = wxMenu()`
`Itemid = integer()`
`Text = chardata() (see module unicode)`

Equivalent to *append(This, Itemid, Text, [])*.

append(This, Itemid, Text, Submenu) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)
Submenu = wxMenu()

See **external documentation**.
Also:
append(This, Itemid, Text, [Option]) -> wxMenuItem:wxMenuItem() when
This::wxMenu(), Itemid::integer(), Text::unicode:chardata(),
Option :: {help, unicode:chardata()}
| {kind, wx:wx_enum()}.

Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

append(This, Itemid, Text, Help, IsCheckable) -> ok

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)
Help = chardata() (see module unicode)
IsCheckable = boolean()

See **external documentation**.
Also:
append(This, Itemid, Text, Submenu, [Option]) -> wxMenuItem:wxMenuItem() when
This::wxMenu(), Itemid::integer(), Text::unicode:chardata(), Submenu::wxMenu(),
Option :: {help, unicode:chardata()}.

appendCheckItem(This, Itemid, Text) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)

Equivalent to *appendCheckItem(This, Itemid, Text, [])*.

appendCheckItem(This, Itemid, Text, Option::[Option]) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)
Option = {help, chardata() (see module unicode)}

See **external documentation**.

appendRadioItem(This, Itemid, Text) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)

Equivalent to *appendRadioItem(This, Itemid, Text, [])*.

appendRadioItem(This, Itemid, Text, Option::[Option]) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)
Option = {help, chardata() (see module unicode)}

See **external documentation**.

appendSeparator(This) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()

See **external documentation**.

break(This) -> ok

Types:

This = wxMenu()

See **external documentation**.

check(This, Itemid, Check) -> ok

Types:

This = wxMenu()
Itemid = integer()
Check = boolean()

See **external documentation**.

delete(This, Itemid) -> boolean()

Types:

This = wxMenu()
Itemid = integer()

See **external documentation**.
Also:
delete(This, Item) -> boolean() when
This::wxMenu(), Item::wxMenuItem:wxMenuItem().

Destroy(This, Itemid) -> boolean()

Types:

```
This = wxMenu()
Itemid = integer()
```

See **external documentation**.
Also:
'Destroy'(This, Item) -> boolean() when
This::wxMenu(), Item::wxMenuItem:wxMenuItem().

enable(This, Itemid, Enable) -> ok

Types:

```
This = wxMenu()
Itemid = integer()
Enable = boolean()
```

See **external documentation**.

findItem(This, Itemid) -> wxMenuItem() (see module wxMenuItem)

Types:

```
This = wxMenu()
Itemid = integer()
```

See **external documentation**.
Also:
findItem(This, Item) -> integer() when
This::wxMenu(), Item::unicode:chardata().

findItemByPosition(This, Position) -> wxMenuItem() (see module wxMenuItem)

Types:

```
This = wxMenu()
Position = integer()
```

See **external documentation**.

getHelpString(This, Itemid) -> charlist() (see module unicode)

Types:

```
This = wxMenu()
Itemid = integer()
```

See **external documentation**.

getLabel(This, Itemid) -> charlist() (see module unicode)

Types:

```
This = wxMenu()
Itemid = integer()
```

See **external documentation**.

getMenuItemCount(This) -> integer()

Types:

```
This = wxMenu()
```

See **external documentation**.

getMenuItems(This) -> [wxMenuItem() (see module wxMenuItem)]

Types:

This = wxMenu()

See **external documentation**.

getTitle(This) -> charlist() (see module unicode)

Types:

This = wxMenu()

See **external documentation**.

insert(This, Pos, Itemid) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()

Pos = integer()

Itemid = integer()

See **external documentation**.
Also:
insert(This, Pos, Item) -> wxMenuItem:wxMenuItem() when
This::wxMenu(), Pos::integer(), Item::wxMenuItem:wxMenuItem().

Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

insert(This, Pos, Itemid, Option::[Option]) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()

Pos = integer()

Itemid = integer()

Option = {text, chardata() (see module unicode)} | {help, chardata() (see module unicode)} | {kind, wx_enum() (see module wx)}

See **external documentation**.
Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

insert(This, Pos, Itemid, Text, Submenu) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()

Pos = integer()

Itemid = integer()

Text = chardata() (see module unicode)

Submenu = wxMenu()

Equivalent to *insert(This, Pos, Itemid, Text, Submenu, [])*.

insert(This, Pos, Itemid, Text, Help, IsCheckable) -> ok

Types:

```
This = wxMenu()
Pos = integer()
Itemid = integer()
Text = chardata() (see module unicode)
Help = chardata() (see module unicode)
IsCheckable = boolean()
```

See **external documentation**.
Also:
insert(This, Pos, Itemid, Text, Submenu, [Option]) -> wxMenuItem:wxMenuItem() when This::wxMenu(), Pos::integer(), Itemid::integer(), Text::unicode:chardata(), Submenu::wxMenu(), Option :: {help, unicode:chardata()}.

insertCheckItem(This, Pos, Itemid, Text) -> wxMenuItem() (see module wxMenuItem)

Types:

```
This = wxMenu()
Pos = integer()
Itemid = integer()
Text = chardata() (see module unicode)
```

Equivalent to *insertCheckItem(This, Pos, Itemid, Text, [])*.

insertCheckItem(This, Pos, Itemid, Text, Option::[Option]) -> wxMenuItem() (see module wxMenuItem)

Types:

```
This = wxMenu()
Pos = integer()
Itemid = integer()
Text = chardata() (see module unicode)
Option = {help, chardata() (see module unicode)}
```

See **external documentation**.

insertRadioItem(This, Pos, Itemid, Text) -> wxMenuItem() (see module wxMenuItem)

Types:

```
This = wxMenu()
Pos = integer()
Itemid = integer()
Text = chardata() (see module unicode)
```

Equivalent to *insertRadioItem(This, Pos, Itemid, Text, [])*.

insertRadioItem(This, Pos, Itemid, Text, Option::[Option]) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Pos = integer()
Itemid = integer()
Text = chardata() (see module unicode)
Option = {help, chardata() (see module unicode)}

See **external documentation**.

insertSeparator(This, Pos) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Pos = integer()

See **external documentation**.

isChecked(This, Itemid) -> boolean()

Types:

This = wxMenu()
Itemid = integer()

See **external documentation**.

isEnabled(This, Itemid) -> boolean()

Types:

This = wxMenu()
Itemid = integer()

See **external documentation**.

prepend(This, Itemid) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()

See **external documentation**.
Also:
prepend(This, Item) -> wxMenuItem:wxMenuItem() when
This::wxMenu(), Item::wxMenuItem:wxMenuItem().

Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

prepend(This, Itemid, Option::[Option]) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()

Option = {text, chardata() (see module unicode)} | {help, chardata() (see module unicode)} | {kind, wx_enum() (see module wx)}

See **external documentation**.
Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

prepend(This, Itemid, Text, Submenu) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)
Submenu = wxMenu()

Equivalent to *prepend(This, Itemid, Text, Submenu, [])*.

prepend(This, Itemid, Text, Help, IsCheckable) -> ok

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)
Help = chardata() (see module unicode)
IsCheckable = boolean()

See **external documentation**.
Also:
prepend(This, Itemid, Text, Submenu, [Option]) -> wxMenuItem:wxMenuItem() when
This::wxMenu(), Itemid::integer(), Text::unicode:chardata(), Submenu::wxMenu(),
Option :: {help, unicode:chardata()}.

prependCheckItem(This, Itemid, Text) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)

Equivalent to *prependCheckItem(This, Itemid, Text, [])*.

prependCheckItem(This, Itemid, Text, Option::[Option]) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)
Option = {help, chardata() (see module unicode)}

See **external documentation**.

prependRadioItem(This, Itemid, Text) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)

Equivalent to *prependRadioItem(This, Itemid, Text, [])*.

prependRadioItem(This, Itemid, Text, Option::[Option]) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()
Text = chardata() (see module unicode)
Option = {help, chardata() (see module unicode)}

See **external documentation**.

prependSeparator(This) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()

See **external documentation**.

remove(This, Itemid) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenu()
Itemid = integer()

See **external documentation**.
Also:
remove(This, Item) -> wxMenuItem:wxMenuItem() when
This::wxMenu(), Item::wxMenuItem:wxMenuItem().

setHelpString(This, Itemid, HelpString) -> ok

Types:

This = wxMenu()
Itemid = integer()
HelpString = chardata() (see module unicode)

See **external documentation**.

setLabel(This, Itemid, Label) -> ok

Types:

This = wxMenu()
Itemid = integer()
Label = chardata() (see module unicode)

See **external documentation**.

setTitle(This, Title) -> ok

Types:

```
This = wxMenu()
Title = chardata() (see module unicode)
```

See **external documentation**.

destroy(This::wxMenu()) -> ok

Destroys this object, do not use object again

wxMenuBar

Erlang module

See external documentation: **wxMenuBar**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxMenuBar()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMenuBar()`

See **external documentation**.

`new(Style) -> wxMenuBar()`

Types:

```
Style = integer()
```

See **external documentation**.

`append(This, Menu, Title) -> boolean()`

Types:

```
This = wxMenuBar()
Menu = wxMenu() (see module wxMenu)
Title = chardata() (see module unicode)
```

See **external documentation**.

`check(This, Itemid, Check) -> ok`

Types:

```
This = wxMenuBar()
Itemid = integer()
Check = boolean()
```

See **external documentation**.

`enable(This) -> boolean()`

Types:

```
This = wxMenuBar()
```

Equivalent to *enable(This, [])*.

enable(This, Option::[Option]) -> boolean()

Types:

This = wxMenuBar()
Option = {enable, boolean()}

See **external documentation**.

enable(This, Itemid, Enable) -> ok

Types:

This = wxMenuBar()
Itemid = integer()
Enable = boolean()

See **external documentation**.

enableTop(This, Pos, Flag) -> ok

Types:

This = wxMenuBar()
Pos = integer()
Flag = boolean()

See **external documentation**.

findMenu(This, Title) -> integer()

Types:

This = wxMenuBar()
Title = chardata() (see module unicode)

See **external documentation**.

findMenuItem(This, MenuString, ItemString) -> integer()

Types:

This = wxMenuBar()
MenuString = chardata() (see module unicode)
ItemString = chardata() (see module unicode)

See **external documentation**.

findItem(This, Id) -> wxMenuItem() (see module wxMenuItem)

Types:

This = wxMenuBar()
Id = integer()

See **external documentation**.

getHelpString(This, Itemid) -> charlist() (see module unicode)

Types:

This = wxMenuBar()
Itemid = integer()

See **external documentation**.

getLabel(This) -> charlist() (see module unicode)

Types:

This = wxMenuBar()

See **external documentation**.

getLabel(This, Itemid) -> charlist() (see module unicode)

Types:

This = wxMenuBar()
Itemid = integer()

See **external documentation**.

getLabelTop(This, Pos) -> charlist() (see module unicode)

Types:

This = wxMenuBar()
Pos = integer()

See **external documentation**.

getMenu(This, Pos) -> wxMenu() (see module wxMenu)

Types:

This = wxMenuBar()
Pos = integer()

See **external documentation**.

getMenuCount(This) -> integer()

Types:

This = wxMenuBar()

See **external documentation**.

insert(This, Pos, Menu, Title) -> boolean()

Types:

This = wxMenuBar()
Pos = integer()
Menu = wxMenu() (see module wxMenu)
Title = chardata() (see module unicode)

See **external documentation**.

isChecked(This, Itemid) -> boolean()

Types:

This = wxMenuBar()
Itemid = integer()

See **external documentation**.

isEnabled(This) -> boolean()

Types:

This = wxMenuBar()

See **external documentation**.

isEnabled(This, Itemid) -> boolean()

Types:

This = wxMenuBar()
Itemid = integer()

See **external documentation**.

remove(This, Pos) -> wxMenu() (see module wxMenu)

Types:

This = wxMenuBar()
Pos = integer()

See **external documentation**.

replace(This, Pos, Menu, Title) -> wxMenu() (see module wxMenu)

Types:

This = wxMenuBar()
Pos = integer()
Menu = wxMenu() (see module wxMenu)
Title = chardata() (see module unicode)

See **external documentation**.

setHelpString(This, Itemid, HelpString) -> ok

Types:

This = wxMenuBar()
Itemid = integer()
HelpString = chardata() (see module unicode)

See **external documentation**.

setLabel(This, S) -> ok

Types:

This = wxMenuBar()
S = chardata() (see module unicode)

See **external documentation**.

setLabel(This, Itemid, Label) -> ok

Types:

This = wxMenuBar()
Itemid = integer()
Label = chardata() (see module unicode)

See **external documentation**.

setLabelTop(This, Pos, Label) -> ok

Types:

```
This = wxMenuBar()
Pos = integer()
Label = chardata() (see module unicode)
```

See **external documentation**.

destroy(This::wxMenuBar()) -> ok

Destroys this object, do not use object again

wxMenuEvent

Erlang module

See external documentation: **wxMenuEvent**.

Use *wxEvtHandler:connect/3* with EventType:

menu_open, *menu_close*, *menu_highlight*

See also the message variant *#wxMenu{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxMenuEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getMenu(This) -> wxMenu() (see module wxMenu)`

Types:

`This = wxMenuEvent()`

See **external documentation**.

`getMenuId(This) -> integer()`

Types:

`This = wxMenuEvent()`

See **external documentation**.

`isPopup(This) -> boolean()`

Types:

`This = wxMenuEvent()`

See **external documentation**.

wxMenuItem

Erlang module

See external documentation: **wxMenuItem**.

DATA TYPES

`wxMenuItem()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMenuItem()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxMenuItem()`

Types:

```
Option = {parentMenu, wxMenu() (see module wxMenu)} | {id, integer()} |
{text, chardata() (see module unicode)} | {help, chardata() (see module
unicode)} | {kind, wx_enum() (see module wx)} | {subMenu, wxMenu() (see
module wxMenu)}
```

See **external documentation**.
Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

`check(This) -> ok`

Types:

```
This = wxMenuItem()
```

Equivalent to *check(This, [])*.

`check(This, Option::[Option]) -> ok`

Types:

```
This = wxMenuItem()
Option = {check, boolean()}
```

See **external documentation**.

`enable(This) -> ok`

Types:

```
This = wxMenuItem()
```

Equivalent to *enable(This, [])*.

`enable(This, Option::[Option]) -> ok`

Types:

```
This = wxMenuItem()
Option = {enable, boolean()}
```

See **external documentation**.

getBitmap(This) -> wxBitmap() (see module wxBitmap)

Types:

```
This = wxMenuItem()
```

See **external documentation**.

getHelp(This) -> charlist() (see module unicode)

Types:

```
This = wxMenuItem()
```

See **external documentation**.

getId(This) -> integer()

Types:

```
This = wxMenuItem()
```

See **external documentation**.

getKind(This) -> wx_enum() (see module wx)

Types:

```
This = wxMenuItem()
```

See **external documentation**.
Res = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

getLabel(This) -> charlist() (see module unicode)

Types:

```
This = wxMenuItem()
```

See **external documentation**.

getLabelFromText(Text) -> charlist() (see module unicode)

Types:

```
Text = chardata() (see module unicode)
```

See **external documentation**.

getMenu(This) -> wxMenu() (see module wxMenu)

Types:

```
This = wxMenuItem()
```

See **external documentation**.

getText(This) -> charlist() (see module unicode)

Types:

```
This = wxMenuItem()
```

See **external documentation**.

getSubMenu(This) -> wxMenu() (see module wxMenu)

Types:

This = wxMenuItem()

See **external documentation**.

isCheckable(This) -> boolean()

Types:

This = wxMenuItem()

See **external documentation**.

isChecked(This) -> boolean()

Types:

This = wxMenuItem()

See **external documentation**.

isEnabled(This) -> boolean()

Types:

This = wxMenuItem()

See **external documentation**.

isSeparator(This) -> boolean()

Types:

This = wxMenuItem()

See **external documentation**.

isSubMenu(This) -> boolean()

Types:

This = wxMenuItem()

See **external documentation**.

setBitmap(This, Bitmap) -> ok

Types:

This = wxMenuItem()

Bitmap = wxBitmap() (see module wxBitmap)

See **external documentation**.

setHelp(This, Str) -> ok

Types:

This = wxMenuItem()

Str = chardata() (see module unicode)

See **external documentation**.

setMenu(This, Menu) -> ok

Types:

This = wxMenuItem()

Menu = wxMenu() (see module wxMenu)

See **external documentation**.

setSubMenu(This, Menu) -> ok

Types:

This = wxMenuItem()

Menu = wxMenu() (see module wxMenu)

See **external documentation**.

setText(This, Str) -> ok

Types:

This = wxMenuItem()

Str = chardata() (see module unicode)

See **external documentation**.

destroy(This::wxMenuItem()) -> ok

Destroys this object, do not use object again

wxMessageDialog

Erlang module

See external documentation: **wxMessageDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxMessageDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent, Message) -> wxMessageDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`
`Message = chardata() (see module unicode)`

Equivalent to *new(Parent, Message, [])*.

`new(Parent, Message, Option::[Option]) -> wxMessageDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`
`Message = chardata() (see module unicode)`
`Option = {caption, chardata() (see module unicode)} | {style, integer()} | {pos, {X::integer(), Y::integer()}}`

See **external documentation**.

`destroy(This::wxMessageDialog()) -> ok`

Destroys this object, do not use object again

wxMiniFrame

Erlang module

See external documentation: **wxMiniFrame**.

This class is derived (and can use functions) from:
wxFrame
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxMiniFrame()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMiniFrame()`

See **external documentation**.

`new(Parent, Id, Title) -> wxMiniFrame()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Title, [])*.

`new(Parent, Id, Title, Option::[Option]) -> wxMiniFrame()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent, Id, Title) -> boolean()`

Types:

```
This = wxMiniFrame()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Title, [])*.

create(This, Parent, Id, Title, Option::[Option]) -> boolean()

Types:

```
This = wxMiniFrame()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

destroy(This::wxMiniFrame()) -> ok

Destroys this object, do not use object again

wxMirrorDC

Erlang module

See external documentation: **wxMirrorDC**.

This class is derived (and can use functions) from:
wxDC

DATA TYPES

`wxMirrorDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Dc, Mirror) -> wxMirrorDC()`

Types:

```
Dc = wxDC() (see module wxDC)
Mirror = boolean()
```

See **external documentation**.

`destroy(This::wxMirrorDC()) -> ok`

Destroys this object, do not use object again

wxMouseCaptureChangedEvent

Erlang module

See external documentation: **wxMouseCaptureChangedEvent**.

Use *wxEvtHandler:connect/3* with EventType:

mouse_capture_changed

See also the message variant *#wxMouseCaptureChanged{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxMouseCaptureChangedEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getCapturedWindow(This) -> wxWindow() (see module wxWindow)`

Types:

`This = wxMouseCaptureChangedEvent()`

See **external documentation**.

wxMouseEvent

Erlang module

See external documentation: **wxMouseEvent**.

Use *wxEvtHandler:connect/3* with EventType:

left_down, *left_up*, *middle_down*, *middle_up*, *right_down*, *right_up*, *motion*, *enter_window*, *leave_window*, *left_dclick*, *middle_dclick*, *right_dclick*, *mousewheel*, *nc_left_down*, *nc_left_up*, *nc_middle_down*, *nc_middle_up*, *nc_right_down*, *nc_right_up*, *nc_motion*, *nc_enter_window*, *nc_leave_window*, *nc_left_dclick*, *nc_middle_dclick*, *nc_right_dclick*

See also the message variant *#wxMouse{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxMouseEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`altDown(This) -> boolean()`

Types:

`This = wxMouseEvent()`

See **external documentation**.

`button(This, But) -> boolean()`

Types:

`This = wxMouseEvent()`
`But = integer()`

See **external documentation**.

`buttonDClick(This) -> boolean()`

Types:

`This = wxMouseEvent()`

Equivalent to *buttonDClick(This, [])*.

`buttonDClick(This, Option::[Option]) -> boolean()`

Types:

`This = wxMouseEvent()`
`Option = {but, integer()}`

See **external documentation**.

buttonDown(This) -> boolean()

Types:

This = wxMouseEvent()

Equivalent to *buttonDown(This, [])*.

buttonDown(This, Option::[Option]) -> boolean()

Types:

This = wxMouseEvent()

Option = {but, integer()}

See **external documentation**.

buttonUp(This) -> boolean()

Types:

This = wxMouseEvent()

Equivalent to *buttonUp(This, [])*.

buttonUp(This, Option::[Option]) -> boolean()

Types:

This = wxMouseEvent()

Option = {but, integer()}

See **external documentation**.

cmdDown(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

controlDown(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

dragging(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

entering(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

getButton(This) -> integer()

Types:

This = wxMouseEvent()

See **external documentation**.

getPosition(This) -> {X::integer(), Y::integer()}

Types:

This = wxMouseEvent()

See **external documentation**.

getLogicalPosition(This, Dc) -> {X::integer(), Y::integer()}

Types:

This = wxMouseEvent()

Dc = wxDC() (see module wxDC)

See **external documentation**.

getLinesPerAction(This) -> integer()

Types:

This = wxMouseEvent()

See **external documentation**.

getWheelRotation(This) -> integer()

Types:

This = wxMouseEvent()

See **external documentation**.

getWheelDelta(This) -> integer()

Types:

This = wxMouseEvent()

See **external documentation**.

getX(This) -> integer()

Types:

This = wxMouseEvent()

See **external documentation**.

getY(This) -> integer()

Types:

This = wxMouseEvent()

See **external documentation**.

isButton(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

isPageScroll(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

leaving(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

leftDClick(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

leftDown(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

leftIsDown(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

leftUp(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

metaDown(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

middleDClick(This) -> boolean()

Types:

```
This = wxMouseEvent()
```

See **external documentation**.

middleDown(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

middleIsDown(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

middleUp(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

moving(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

rightDClick(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

rightDown(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

rightIsDown(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

rightUp(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

shiftDown(This) -> boolean()

Types:

This = wxMouseEvent()

See **external documentation**.

wxMoveEvent

Erlang module

See external documentation: **wxMoveEvent**.

Use *wxEvtHandler:connect/3* with EventType:

move

See also the message variant *#wxMove{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxMoveEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getPosition(This) -> {X::integer(), Y::integer()}`

Types:

`This = wxMoveEvent()`

See **external documentation**.

wxMultiChoiceDialog

Erlang module

See external documentation: **wxMultiChoiceDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxMultiChoiceDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxMultiChoiceDialog()`

See **external documentation**.

`new(Parent, Message, Caption, Choices) -> wxMultiChoiceDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Message = chardata() (see module unicode)
Caption = chardata() (see module unicode)
Choices = [chardata() (see module unicode)]
```

Equivalent to *new(Parent, Message, Caption, Choices, [])*.

`new(Parent, Message, Caption, Choices, Option::[Option]) -> wxMultiChoiceDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Message = chardata() (see module unicode)
Caption = chardata() (see module unicode)
Choices = [chardata() (see module unicode)]
Option = {style, integer()} | {pos, {X::integer(), Y::integer()}}
```

See **external documentation**.

`getSelections(This) -> [integer()]`

Types:

```
This = wxMultiChoiceDialog()
```

See **external documentation**.

setSelections(This, Selections) -> ok

Types:

```
This = wxMultiChoiceDialog()
Selections = [integer()]
```

See **external documentation**.

destroy(This::wxMultiChoiceDialog()) -> ok

Destroys this object, do not use object again

wxNavigationKeyEvent

Erlang module

See external documentation: **wxNavigationKeyEvent**.

Use *wxEvtHandler:connect/3* with EventType:

navigation_key

See also the message variant *#wxNavigationKey{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxNavigationKeyEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getDirection(This) -> boolean()`

Types:

`This = wxNavigationKeyEvent()`

See **external documentation**.

`setDirection(This, BForward) -> ok`

Types:

`This = wxNavigationKeyEvent()`
`BForward = boolean()`

See **external documentation**.

`isWindowChange(This) -> boolean()`

Types:

`This = wxNavigationKeyEvent()`

See **external documentation**.

`setWindowChange(This, BIs) -> ok`

Types:

`This = wxNavigationKeyEvent()`
`BIs = boolean()`

See **external documentation**.

`isFromTab(This) -> boolean()`

Types:

`This = wxNavigationKeyEvent()`

See **external documentation**.

setFromTab(This, BIs) -> ok

Types:

This = wxNavigationKeyEvent()

BIs = boolean()

See **external documentation**.

getCurrentFocus(This) -> wxWindow() (see module wxWindow)

Types:

This = wxNavigationKeyEvent()

See **external documentation**.

setCurrentFocus(This, Win) -> ok

Types:

This = wxNavigationKeyEvent()

Win = wxWindow() (see module wxWindow)

See **external documentation**.

wxNcPaintEvent

Erlang module

See external documentation: **wxNcPaintEvent**.

Use *wxEvtHandler:connect/3* with EventType:

nc_paint

See also the message variant *#wxNcPaint{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxNcPaintEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxNotebook

Erlang module

See external documentation: **wxNotebook**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxNotebook()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxNotebook()`

See **external documentation**.

`new(Parent, Winid) -> wxNotebook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Winid = integer()
```

Equivalent to *new(Parent, Winid, [])*.

`new(Parent, Winid, Option::[Option]) -> wxNotebook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Winid = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`addPage(This, Page, Text) -> boolean()`

Types:

```
This = wxNotebook()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
```

Equivalent to *addPage(This, Page, Text, [])*.

`addPage(This, Page, Text, Option::[Option]) -> boolean()`

Types:

```
This = wxNotebook()
```

```
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}
```

See **external documentation**.

advanceSelection(This) -> ok

Types:

```
This = wxNotebook()
```

Equivalent to *advanceSelection(This, [])*.

advanceSelection(This, Option::[Option]) -> ok

Types:

```
This = wxNotebook()
Option = {forward, boolean()}
```

See **external documentation**.

assignImageList(This, ImageList) -> ok

Types:

```
This = wxNotebook()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

create(This, Parent, Id) -> boolean()

Types:

```
This = wxNotebook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

create(This, Parent, Id, Option::[Option]) -> boolean()

Types:

```
This = wxNotebook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

deleteAllPages(This) -> boolean()

Types:

```
This = wxNotebook()
```

See **external documentation**.

deletePage(This, NPage) -> boolean()

Types:

This = wxNotebook()
NPage = integer()

See **external documentation**.

removePage(This, NPage) -> boolean()

Types:

This = wxNotebook()
NPage = integer()

See **external documentation**.

getCurrentPage(This) -> wxWindow() (see module wxWindow)

Types:

This = wxNotebook()

See **external documentation**.

getImageList(This) -> wxImageList() (see module wxImageList)

Types:

This = wxNotebook()

See **external documentation**.

getPage(This, N) -> wxWindow() (see module wxWindow)

Types:

This = wxNotebook()
N = integer()

See **external documentation**.

getPageCount(This) -> integer()

Types:

This = wxNotebook()

See **external documentation**.

getPageImage(This, NPage) -> integer()

Types:

This = wxNotebook()
NPage = integer()

See **external documentation**.

getPageText(This, NPage) -> charlist() (see module unicode)

Types:

This = wxNotebook()
NPage = integer()

See **external documentation**.

getRowCount(This) -> integer()

Types:

This = wxNotebook()

See **external documentation**.

getSelection(This) -> integer()

Types:

This = wxNotebook()

See **external documentation**.

getThemeBackgroundColour(This) -> wx_colour4() (see module wx)

Types:

This = wxNotebook()

See **external documentation**.

hitTest(This, Pt) -> Result

Types:

Result = {Res::integer(), Flags::integer()}
This = wxNotebook()
Pt = {X::integer(), Y::integer()}

See **external documentation**.

insertPage(This, Position, Win, StrText) -> boolean()

Types:

This = wxNotebook()
Position = integer()
Win = wxWindow() (see module wxWindow)
StrText = chardata() (see module unicode)

Equivalent to *insertPage(This, Position, Win, StrText, [])*.

insertPage(This, Position, Win, StrText, Option::[Option]) -> boolean()

Types:

This = wxNotebook()
Position = integer()
Win = wxWindow() (see module wxWindow)
StrText = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}

See **external documentation**.

setImageList(This, ImageList) -> ok

Types:

```
This = wxNotebook()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

setPadding(This, Padding) -> ok

Types:

```
This = wxNotebook()
Padding = {W::integer(), H::integer()}
```

See **external documentation**.

setPageSize(This, Size) -> ok

Types:

```
This = wxNotebook()
Size = {W::integer(), H::integer()}
```

See **external documentation**.

setPageImage(This, NPage, NImage) -> boolean()

Types:

```
This = wxNotebook()
NPage = integer()
NImage = integer()
```

See **external documentation**.

setPageText(This, NPage, StrText) -> boolean()

Types:

```
This = wxNotebook()
NPage = integer()
StrText = chardata() (see module unicode)
```

See **external documentation**.

setSelection(This, NPage) -> integer()

Types:

```
This = wxNotebook()
NPage = integer()
```

See **external documentation**.

changeSelection(This, NPage) -> integer()

Types:

```
This = wxNotebook()
NPage = integer()
```

See **external documentation**.

destroy(This::wxNotebook()) -> ok

Destroys this object, do not use object again

wxNotebookEvent

Erlang module

See external documentation: **wxNotebookEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_notebook_page_changed, *command_notebook_page_changing*

See also the message variant *#wxNotebook{}* event record type.

This class is derived (and can use functions) from:
wxNotifyEvent
wxCommandEvent
wxEvent

DATA TYPES

`wxNotebookEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getOldSelection(This) -> integer()`

Types:

`This = wxNotebookEvent()`

See **external documentation**.

`getSelection(This) -> integer()`

Types:

`This = wxNotebookEvent()`

See **external documentation**.

`setOldSelection(This, NOldSel) -> ok`

Types:

`This = wxNotebookEvent()`
`NOldSel = integer()`

See **external documentation**.

`setSelection(This, NSel) -> ok`

Types:

`This = wxNotebookEvent()`
`NSel = integer()`

See **external documentation**.

wxNotifyEvent

Erlang module

See external documentation: **wxNotifyEvent**.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxNotifyEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`allow(This) -> ok`

Types:

`This = wxNotifyEvent()`

See **external documentation**.

`isAllowed(This) -> boolean()`

Types:

`This = wxNotifyEvent()`

See **external documentation**.

`veto(This) -> ok`

Types:

`This = wxNotifyEvent()`

See **external documentation**.

wxPageSetupDialog

Erlang module

See external documentation: **wxPageSetupDialog**.

DATA TYPES

`wxPageSetupDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent) -> wxPageSetupDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxPageSetupDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`

`Option = {data, wxPageSetupDialogData() (see module wxPageSetupDialogData)}`

See **external documentation**.

`getPageSetupData(This) -> wxPageSetupDialogData() (see module wxPageSetupDialogData)`

Types:

`This = wxPageSetupDialog()`

See **external documentation**.

`showModal(This) -> integer()`

Types:

`This = wxPageSetupDialog()`

See **external documentation**.

`destroy(This::wxPageSetupDialog()) -> ok`

Destroys this object, do not use object again

wxPageSetupDialogData

Erlang module

See external documentation: **wxPageSetupDialogData**.

DATA TYPES

`wxPageSetupDialogData()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPageSetupDialogData()`

See **external documentation**.

`new(PrintData) -> wxPageSetupDialogData()`

Types:

```
PrintData = wxPrintData() (see module wxPrintData) |
wxPageSetupDialogData()
```

See **external documentation**.

`enableHelp(This, Flag) -> ok`

Types:

```
This = wxPageSetupDialogData()
Flag = boolean()
```

See **external documentation**.

`enableMargins(This, Flag) -> ok`

Types:

```
This = wxPageSetupDialogData()
Flag = boolean()
```

See **external documentation**.

`enableOrientation(This, Flag) -> ok`

Types:

```
This = wxPageSetupDialogData()
Flag = boolean()
```

See **external documentation**.

`enablePaper(This, Flag) -> ok`

Types:

```
This = wxPageSetupDialogData()
```

```
Flag = boolean()
```

See **external documentation**.

enablePrinter(This, Flag) -> ok

Types:

```
This = wxPageSetupDialogData()
Flag = boolean()
```

See **external documentation**.

getDefaultMinMargins(This) -> boolean()

Types:

```
This = wxPageSetupDialogData()
```

See **external documentation**.

getEnableMargins(This) -> boolean()

Types:

```
This = wxPageSetupDialogData()
```

See **external documentation**.

getEnableOrientation(This) -> boolean()

Types:

```
This = wxPageSetupDialogData()
```

See **external documentation**.

getEnablePaper(This) -> boolean()

Types:

```
This = wxPageSetupDialogData()
```

See **external documentation**.

getEnablePrinter(This) -> boolean()

Types:

```
This = wxPageSetupDialogData()
```

See **external documentation**.

getEnableHelp(This) -> boolean()

Types:

```
This = wxPageSetupDialogData()
```

See **external documentation**.

getDefaultInfo(This) -> boolean()

Types:

```
This = wxPageSetupDialogData()
```

See **external documentation**.

getMarginTopLeft(This) -> {X::integer(), Y::integer()}

Types:

This = wxPageSetupDialogData()

See **external documentation**.

getMarginBottomRight(This) -> {X::integer(), Y::integer()}

Types:

This = wxPageSetupDialogData()

See **external documentation**.

getMinMarginTopLeft(This) -> {X::integer(), Y::integer()}

Types:

This = wxPageSetupDialogData()

See **external documentation**.

getMinMarginBottomRight(This) -> {X::integer(), Y::integer()}

Types:

This = wxPageSetupDialogData()

See **external documentation**.

getPaperId(This) -> integer()

Types:

This = wxPageSetupDialogData()

See **external documentation**.

getPaperSize(This) -> {W::integer(), H::integer()}

Types:

This = wxPageSetupDialogData()

See **external documentation**.

getPrintData(This) -> wxPrintData() (see module wxPrintData)

Types:

This = wxPageSetupDialogData()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxPageSetupDialogData()

See **external documentation**.

setDefaultInfo(This, Flag) -> ok

Types:

This = wxPageSetupDialogData()

```
Flag = boolean()
```

See **external documentation**.

setDefaultMinMargins(This, Flag) -> ok

Types:

```
This = wxPageSetupDialogData()
Flag = boolean()
```

See **external documentation**.

setMarginTopLeft(This, Pt) -> ok

Types:

```
This = wxPageSetupDialogData()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

setMarginBottomRight(This, Pt) -> ok

Types:

```
This = wxPageSetupDialogData()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

setMinMarginTopLeft(This, Pt) -> ok

Types:

```
This = wxPageSetupDialogData()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

setMinMarginBottomRight(This, Pt) -> ok

Types:

```
This = wxPageSetupDialogData()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

setPaperId(This, Id) -> ok

Types:

```
This = wxPageSetupDialogData()
Id = integer()
```

See **external documentation**.

setPaperSize(This, Id) -> ok

Types:

```
This = wxPageSetupDialogData()
Id = integer()
```

See **external documentation**.
Also:
setPaperSize(This, Sz) -> ok when
This::wxPageSetupDialogData(), Sz::{W::integer(), H::integer()}.

`setPrintData(This, PrintData) -> ok`

Types:

```
This = wxPageSetupDialogData()
PrintData = wxPrintData() (see module wxPrintData)
```

See **external documentation**.

`destroy(This::wxPageSetupDialogData()) -> ok`

Destroys this object, do not use object again

wxPaintDC

Erlang module

See external documentation: **wxPaintDC**.

This class is derived (and can use functions) from:
wxWindowDC
wxDC

DATA TYPES

`wxPaintDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPaintDC()`

See **external documentation**.

`new(Win) -> wxPaintDC()`

Types:

`Win = wxWindow() (see module wxWindow)`

See **external documentation**.

`destroy(This::wxPaintDC()) -> ok`

Destroys this object, do not use object again

wxPaintEvent

Erlang module

See external documentation: **wxPaintEvent**.

Use *wxEvtHandler:connect/3* with EventType:

paint, *paint_icon*

See also the message variant *#wxPaint{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxPaintEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxPalette

Erlang module

See external documentation: **wxPalette**.

DATA TYPES

`wxPalette()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPalette()`

See **external documentation**.

`new(Red, Green, Blue) -> wxPalette()`

Types:

```
Red = binary()
Green = binary()
Blue = binary()
```

See **external documentation**.

`create(This, Red, Green, Blue) -> boolean()`

Types:

```
This = wxPalette()
Red = binary()
Green = binary()
Blue = binary()
```

See **external documentation**.

`getColoursCount(This) -> integer()`

Types:

```
This = wxPalette()
```

See **external documentation**.

`getPixel(This, Red, Green, Blue) -> integer()`

Types:

```
This = wxPalette()
Red = integer()
Green = integer()
Blue = integer()
```

See **external documentation**.

getRGB(This, Pixel) -> Result

Types:

```
Result = {Res::boolean(), Red::integer(), Green::integer(),
Blue::integer()}
This = wxPalette()
Pixel = integer()
```

See **external documentation**.

isOk(This) -> boolean()

Types:

```
This = wxPalette()
```

See **external documentation**.

destroy(This::wxPalette()) -> ok

Destroys this object, do not use object again

wxPaletteChangedEvent

Erlang module

See external documentation: **wxPaletteChangedEvent**.

Use *wxEvtHandler:connect/3* with EventType:

palette_changed

See also the message variant *#wxPaletteChanged{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxPaletteChangedEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`setChangedWindow(This, Win) -> ok`

Types:

```
This = wxPaletteChangedEvent()
Win = wxWindow() (see module wxWindow)
```

See **external documentation**.

`getChangedWindow(This) -> wxWindow() (see module wxWindow)`

Types:

```
This = wxPaletteChangedEvent()
```

See **external documentation**.

wxPanel

Erlang module

See external documentation: **wxPanel**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxPanel()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPanel()`

See **external documentation**.

`new(Parent) -> wxPanel()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxPanel()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {winid, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`new(Parent, X, Y, Width, Height) -> wxPanel()`

Types:

```
Parent = wxWindow() (see module wxWindow)
X = integer()
Y = integer()
Width = integer()
Height = integer()
```

Equivalent to *new(Parent, X, Y, Width, Height, [])*.

`new(Parent, X, Y, Width, Height, Option::[Option]) -> wxPanel()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

```
X = integer()
Y = integer()
Width = integer()
Height = integer()
Option = {style, integer()}
```

See **external documentation**.

initDialog(This) -> ok

Types:

```
This = wxPanel()
```

See **external documentation**.

destroy(This::wxPanel()) -> ok

Destroys this object, do not use object again

wxPasswordEntryDialog

Erlang module

See external documentation: **wxPasswordEntryDialog**.

This class is derived (and can use functions) from:
wxTextEntryDialog
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxPasswordEntryDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent, Message) -> wxPasswordEntryDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Message = chardata() (see module unicode)
```

Equivalent to *new(Parent, Message, [])*.

`new(Parent, Message, Option::[Option]) -> wxPasswordEntryDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Message = chardata() (see module unicode)
Option = {caption, chardata() (see module unicode)} | {value, chardata()
(see module unicode)} | {style, integer()} | {pos, {X::integer(),
Y::integer()}}
```

See **external documentation**.

`destroy(This::wxPasswordEntryDialog()) -> ok`

Destroys this object, do not use object again

wxPen

Erlang module

See external documentation: **wxPen**.

DATA TYPES

`wxPen()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPen()`

See **external documentation**.

`new(Colour) -> wxPen()`

Types:

`Colour = wx_colour() (see module wx)`

Equivalent to *new(Colour, [])*.

`new(Colour, Option::[Option]) -> wxPen()`

Types:

`Colour = wx_colour() (see module wx)`

`Option = {width, integer()} | {style, integer()}`

See **external documentation**.

`getCap(This) -> integer()`

Types:

`This = wxPen()`

See **external documentation**.

`getColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxPen()`

See **external documentation**.

`getJoin(This) -> integer()`

Types:

`This = wxPen()`

See **external documentation**.

getStyle(This) -> integer()

Types:

This = wxPen()

See **external documentation**.

getWidth(This) -> integer()

Types:

This = wxPen()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxPen()

See **external documentation**.

setCap(This, CapStyle) -> ok

Types:

This = wxPen()
CapStyle = integer()

See **external documentation**.

setColour(This, Colour) -> ok

Types:

This = wxPen()
Colour = wx_colour() (see module wx)

See **external documentation**.

setColour(This, Red, Green, Blue) -> ok

Types:

This = wxPen()
Red = integer()
Green = integer()
Blue = integer()

See **external documentation**.

setJoin(This, JoinStyle) -> ok

Types:

This = wxPen()
JoinStyle = integer()

See **external documentation**.

setStyle(This, Style) -> ok

Types:

```
This = wxPen()
Style = integer()
```

See **external documentation**.

setWidth(This, Width) -> ok

Types:

```
This = wxPen()
Width = integer()
```

See **external documentation**.

destroy(This::wxPen()) -> ok

Destroys this object, do not use object again

wxPickerBase

Erlang module

See external documentation: **wxPickerBase**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxPickerBase()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`setInternalMargin(This, Newmargin) -> ok`

Types:

`This = wxPickerBase()`
`Newmargin = integer()`

See **external documentation**.

`getInternalMargin(This) -> integer()`

Types:

`This = wxPickerBase()`

See **external documentation**.

`setTextCtrlProportion(This, Prop) -> ok`

Types:

`This = wxPickerBase()`
`Prop = integer()`

See **external documentation**.

`setPickerCtrlProportion(This, Prop) -> ok`

Types:

`This = wxPickerBase()`
`Prop = integer()`

See **external documentation**.

`getTextCtrlProportion(This) -> integer()`

Types:

`This = wxPickerBase()`

See **external documentation**.

getPickerCtrlProportion(This) -> integer()

Types:

This = wxPickerBase()

See **external documentation**.

hasTextCtrl(This) -> boolean()

Types:

This = wxPickerBase()

See **external documentation**.

getTextCtrl(This) -> wxTextCtrl() (see module wxTextCtrl)

Types:

This = wxPickerBase()

See **external documentation**.

isTextCtrlGrowable(This) -> boolean()

Types:

This = wxPickerBase()

See **external documentation**.

setPickerCtrlGrowable(This) -> ok

Types:

This = wxPickerBase()

Equivalent to *setPickerCtrlGrowable(This, [])*.

setPickerCtrlGrowable(This, Option::[Option]) -> ok

Types:

This = wxPickerBase()

Option = {grow, boolean()}

See **external documentation**.

setTextCtrlGrowable(This) -> ok

Types:

This = wxPickerBase()

Equivalent to *setTextCtrlGrowable(This, [])*.

setTextCtrlGrowable(This, Option::[Option]) -> ok

Types:

This = wxPickerBase()

Option = {grow, boolean()}

See **external documentation**.

isPickerCtrlGrowable(This) -> boolean()

Types:

This = wxPickerBase()

See **external documentation**.

wxPostScriptDC

Erlang module

See external documentation: **wxPostScriptDC**.

This class is derived (and can use functions) from:
wxDC

DATA TYPES

`wxPostScriptDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPostScriptDC()`

See **external documentation**.

`new(PrintData) -> wxPostScriptDC()`

Types:

`PrintData = wxPrintData() (see module wxPrintData)`

See **external documentation**.

`setResolution(Ppi) -> ok`

Types:

`Ppi = integer()`

See **external documentation**.

`getResolution() -> integer()`

See **external documentation**.

`destroy(This::wxPostScriptDC()) -> ok`

Destroys this object, do not use object again

wxPreviewCanvas

Erlang module

See external documentation: **wxPreviewCanvas**.

This class is derived (and can use functions) from:
wxScrolledWindow
wxPanel
wxWindow
wxEvtHandler

DATA TYPES

`wxPreviewCanvas()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxPreviewControlBar

Erlang module

See external documentation: **wxPreviewControlBar**.

This class is derived (and can use functions) from:
wxPanel
wxWindow
wxEvtHandler

DATA TYPES

`wxPreviewControlBar()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Preview, Buttons, Parent) -> wxPreviewControlBar()`

Types:

`Preview = wxPrintPreview() (see module wxPrintPreview)`
`Buttons = integer()`
`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Preview, Buttons, Parent, [])*.

`new(Preview, Buttons, Parent, Option::[Option]) -> wxPreviewControlBar()`

Types:

`Preview = wxPrintPreview() (see module wxPrintPreview)`
`Buttons = integer()`
`Parent = wxWindow() (see module wxWindow)`
`Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}`

See **external documentation**.

`createButtons(This) -> ok`

Types:

`This = wxPreviewControlBar()`

See **external documentation**.

`getPrintPreview(This) -> wxPrintPreview() (see module wxPrintPreview)`

Types:

`This = wxPreviewControlBar()`

See **external documentation**.

getZoomControl(This) -> integer()

Types:

This = wxPreviewControlBar()

See **external documentation**.

setZoomControl(This, Zoom) -> ok

Types:

This = wxPreviewControlBar()

Zoom = integer()

See **external documentation**.

destroy(This::wxPreviewControlBar()) -> ok

Destroys this object, do not use object again

wxPreviewFrame

Erlang module

See external documentation: **wxPreviewFrame**.

This class is derived (and can use functions) from:
wxFrame
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxPreviewFrame()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Preview, Parent) -> wxPreviewFrame()`

Types:

```
Preview = wxPrintPreview() (see module wxPrintPreview)
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Preview, Parent, [])*.

`new(Preview, Parent, Option::[Option]) -> wxPreviewFrame()`

Types:

```
Preview = wxPrintPreview() (see module wxPrintPreview)
Parent = wxWindow() (see module wxWindow)
Option = {title, chardata() (see module unicode)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`createControlBar(This) -> ok`

Types:

```
This = wxPreviewFrame()
```

See **external documentation**.

`createCanvas(This) -> ok`

Types:

```
This = wxPreviewFrame()
```

See **external documentation**.

`initialize(This) -> ok`

Types:

```
This = wxPreviewFrame()
```

See **external documentation**.

onCloseWindow(This, Event) -> ok

Types:

```
This = wxPreviewFrame()
Event = wxCloseEvent() (see module wxCloseEvent)
```

See **external documentation**.

destroy(This::wxPreviewFrame()) -> ok

Destroys this object, do not use object again

wxPrintData

Erlang module

See external documentation: **wxPrintData**.

DATA TYPES

`wxPrintData()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPrintData()`

See **external documentation**.

`new(PrintData) -> wxPrintData()`

Types:

`PrintData = wxPrintData()`

See **external documentation**.

`getCollate(This) -> boolean()`

Types:

`This = wxPrintData()`

See **external documentation**.

`getBin(This) -> wx_enum() (see module wx)`

Types:

`This = wxPrintData()`

See **external documentation**.
Res = ?wxPRINTBIN_DEFAULT | ?wxPRINTBIN_ONLYONE | ?wxPRINTBIN_LOWER | ?wxPRINTBIN_MIDDLE | ?wxPRINTBIN_MANUAL | ?wxPRINTBIN_ENVELOPE | ?wxPRINTBIN_ENVMANUAL | ?wxPRINTBIN_AUTO | ?wxPRINTBIN_TRACTOR | ?wxPRINTBIN_SMALLFMT | ?wxPRINTBIN_LARGEFMT | ?wxPRINTBIN_LARGECAPACITY | ?wxPRINTBIN_CASSETTE | ?wxPRINTBIN_FORMSOURCE | ?wxPRINTBIN_USER

`getColour(This) -> boolean()`

Types:

`This = wxPrintData()`

See **external documentation**.

`getDuplex(This) -> wx_enum() (see module wx)`

Types:

`This = wxPrintData()`

See **external documentation**.
Res = ?wxDUPLEX_SIMPLEX | ?wxDUPLEX_HORIZONTAL | ?wxDUPLEX_VERTICAL

getNoCopies(This) -> integer()

Types:

This = wxPrintData()

See **external documentation**.

getOrientation(This) -> integer()

Types:

This = wxPrintData()

See **external documentation**.

getPaperId(This) -> integer()

Types:

This = wxPrintData()

See **external documentation**.

getPrinterName(This) -> charlist() (see module unicode)

Types:

This = wxPrintData()

See **external documentation**.

getQuality(This) -> integer()

Types:

This = wxPrintData()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxPrintData()

See **external documentation**.

setBin(This, Bin) -> ok

Types:

This = wxPrintData()

Bin = wx_enum() (see module wx)

See **external documentation**.
Bin = ?wxPRINTBIN_DEFAULT | ?wxPRINTBIN_ONLYONE | ?wxPRINTBIN_LOWER | ?wxPRINTBIN_MIDDLE | ?wxPRINTBIN_MANUAL | ?wxPRINTBIN_ENVELOPE | ?wxPRINTBIN_ENVMANUAL | ?wxPRINTBIN_AUTO | ?wxPRINTBIN_TRACTOR | ?wxPRINTBIN_SMALLFMT | ?wxPRINTBIN_LARGEFMT | ?wxPRINTBIN_LARGECAPACITY | ?wxPRINTBIN_CASSETTE | ?wxPRINTBIN_FORMSOURCE | ?wxPRINTBIN_USER

setCollate(This, Flag) -> ok

Types:

This = wxPrintData()

Flag = boolean()

See **external documentation**.

setColour(This, Colour) -> ok

Types:

This = wxPrintData()

Colour = boolean()

See **external documentation**.

setDuplex(This, Duplex) -> ok

Types:

This = wxPrintData()

Duplex = wx_enum() (see module wx)

See **external documentation**.

Duplex = ?wxDUPLEX_SIMPLEX | ?wxDUPLEX_HORIZONTAL | ?wxDUPLEX_VERTICAL

setNoCopies(This, V) -> ok

Types:

This = wxPrintData()

V = integer()

See **external documentation**.

setOrientation(This, Orient) -> ok

Types:

This = wxPrintData()

Orient = integer()

See **external documentation**.

setPaperId(This, SizeId) -> ok

Types:

This = wxPrintData()

SizeId = integer()

See **external documentation**.

setPrinterName(This, Name) -> ok

Types:

This = wxPrintData()

Name = chardata() (see module unicode)

See **external documentation**.

setQuality(This, Quality) -> ok

Types:

```
This = wxPrintData()
Quality = integer()
```

See **external documentation**.

destroy(This::wxPrintData()) -> ok

Destroys this object, do not use object again

wxPrintDialog

Erlang module

See external documentation: **wxPrintDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxPrintDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent) -> wxPrintDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxPrintDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`
`Option = {data, wxPrintDialogData() (see module wxPrintDialogData)}`

See **external documentation**.
Also:
new(Parent, Data) -> wxPrintDialog() when
Parent::wxWindow:wxWindow(), Data::wxPrintData:wxPrintData().

`getPrintDialogData(This) -> wxPrintDialogData() (see module wxPrintDialogData)`

Types:

`This = wxPrintDialog()`

See **external documentation**.

`getPrintDC(This) -> wxDC() (see module wxDC)`

Types:

`This = wxPrintDialog()`

See **external documentation**.

destroy(This::wxPrintDialog()) -> ok

Destroys this object, do not use object again

wxPrintDialogData

Erlang module

See external documentation: **wxPrintDialogData**.

DATA TYPES

`wxPrintDialogData()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPrintDialogData()`

See **external documentation**.

`new(DialogData) -> wxPrintDialogData()`

Types:

`DialogData = wxPrintDialogData() | wxPrintData() (see module wxPrintData)`

See **external documentation**.

`enableHelp(This, Flag) -> ok`

Types:

`This = wxPrintDialogData()`
`Flag = boolean()`

See **external documentation**.

`enablePageNumbers(This, Flag) -> ok`

Types:

`This = wxPrintDialogData()`
`Flag = boolean()`

See **external documentation**.

`enablePrintToFile(This, Flag) -> ok`

Types:

`This = wxPrintDialogData()`
`Flag = boolean()`

See **external documentation**.

`enableSelection(This, Flag) -> ok`

Types:

`This = wxPrintDialogData()`
`Flag = boolean()`

See **external documentation**.

getAllPages(This) -> boolean()

Types:

This = wxPrintDialogData()

See **external documentation**.

getCollate(This) -> boolean()

Types:

This = wxPrintDialogData()

See **external documentation**.

getFromPage(This) -> integer()

Types:

This = wxPrintDialogData()

See **external documentation**.

getMaxPage(This) -> integer()

Types:

This = wxPrintDialogData()

See **external documentation**.

getMinPage(This) -> integer()

Types:

This = wxPrintDialogData()

See **external documentation**.

getNoCopies(This) -> integer()

Types:

This = wxPrintDialogData()

See **external documentation**.

getPrintData(This) -> wxPrintData() (see module wxPrintData)

Types:

This = wxPrintDialogData()

See **external documentation**.

getPrintToFile(This) -> boolean()

Types:

This = wxPrintDialogData()

See **external documentation**.

getSelection(This) -> boolean()

Types:

This = wxPrintDialogData()

See **external documentation**.

getToPage(This) -> integer()

Types:

This = wxPrintDialogData()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxPrintDialogData()

See **external documentation**.

setCollate(This, Flag) -> ok

Types:

This = wxPrintDialogData()

Flag = boolean()

See **external documentation**.

setFromPage(This, V) -> ok

Types:

This = wxPrintDialogData()

V = integer()

See **external documentation**.

setMaxPage(This, V) -> ok

Types:

This = wxPrintDialogData()

V = integer()

See **external documentation**.

setMinPage(This, V) -> ok

Types:

This = wxPrintDialogData()

V = integer()

See **external documentation**.

setNoCopies(This, V) -> ok

Types:

This = wxPrintDialogData()

V = integer()

See **external documentation**.

setPrintData(This, PrintData) -> ok

Types:

This = wxPrintDialogData()

PrintData = wxPrintData() (see module wxPrintData)

See **external documentation**.

setPrintToFile(This, Flag) -> ok

Types:

This = wxPrintDialogData()

Flag = boolean()

See **external documentation**.

setSelection(This, Flag) -> ok

Types:

This = wxPrintDialogData()

Flag = boolean()

See **external documentation**.

setToPage(This, V) -> ok

Types:

This = wxPrintDialogData()

V = integer()

See **external documentation**.

destroy(This::wxPrintDialogData()) -> ok

Destroys this object, do not use object again

wxPrintPreview

Erlang module

See external documentation: **wxPrintPreview**.

DATA TYPES

`wxPrintPreview()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Printout) -> wxPrintPreview()`

Types:

`Printout = wxPrintout() (see module wxPrintout)`

Equivalent to *new(Printout, [])*.

`new(Printout, Option::[Option]) -> wxPrintPreview()`

Types:

`Printout = wxPrintout() (see module wxPrintout)`

`Option = {printoutForPrinting, wxPrintout() (see module wxPrintout)} | {data, wxPrintDialogData() (see module wxPrintDialogData)}`

See **external documentation**.

`new(Printout, PrintoutForPrinting, Data) -> wxPrintPreview()`

Types:

`Printout = wxPrintout() (see module wxPrintout)`

`PrintoutForPrinting = wxPrintout() (see module wxPrintout)`

`Data = wxPrintData() (see module wxPrintData)`

See **external documentation**.

`getCanvas(This) -> wxPreviewCanvas() (see module wxPreviewCanvas)`

Types:

`This = wxPrintPreview()`

See **external documentation**.

`getCurrentPage(This) -> integer()`

Types:

`This = wxPrintPreview()`

See **external documentation**.

getFrame(This) -> wxFrame() (see module wxFrame)

Types:

This = wxPrintPreview()

See **external documentation**.

getMaxPage(This) -> integer()

Types:

This = wxPrintPreview()

See **external documentation**.

getMinPage(This) -> integer()

Types:

This = wxPrintPreview()

See **external documentation**.

getPrintout(This) -> wxPrintout() (see module wxPrintout)

Types:

This = wxPrintPreview()

See **external documentation**.

getPrintoutForPrinting(This) -> wxPrintout() (see module wxPrintout)

Types:

This = wxPrintPreview()

See **external documentation**.

isOk(This) -> boolean()

Types:

This = wxPrintPreview()

See **external documentation**.

paintPage(This, Canvas, Dc) -> boolean()

Types:

This = wxPrintPreview()
Canvas = wxPreviewCanvas() (see module wxPreviewCanvas)
Dc = wxDC() (see module wxDC)

See **external documentation**.

print(This, Interactive) -> boolean()

Types:

This = wxPrintPreview()
Interactive = boolean()

See **external documentation**.

renderPage(This, PageNum) -> boolean()

Types:

This = wxPrintPreview()

PageNum = integer()

See **external documentation**.

setCanvas(This, Canvas) -> ok

Types:

This = wxPrintPreview()

Canvas = wxPreviewCanvas() (see module wxPreviewCanvas)

See **external documentation**.

setCurrentPage(This, PageNum) -> boolean()

Types:

This = wxPrintPreview()

PageNum = integer()

See **external documentation**.

setFrame(This, Frame) -> ok

Types:

This = wxPrintPreview()

Frame = wxFrame() (see module wxFrame)

See **external documentation**.

setPrintout(This, Printout) -> ok

Types:

This = wxPrintPreview()

Printout = wxPrintout() (see module wxPrintout)

See **external documentation**.

setZoom(This, Percent) -> ok

Types:

This = wxPrintPreview()

Percent = integer()

See **external documentation**.

destroy(This::wxPrintPreview()) -> ok

Destroys this object, do not use object again

wxPrinter

Erlang module

See external documentation: **wxPrinter**.

DATA TYPES

`wxPrinter()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxPrinter()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxPrinter()`

Types:

`Option = {data, wxPrintDialogData() (see module wxPrintDialogData)}`

See **external documentation**.

`createAbortWindow(This, Parent, Printout) -> wxWindow() (see module wxWindow)`

Types:

`This = wxPrinter()`
`Parent = wxWindow() (see module wxWindow)`
`Printout = wxPrintout() (see module wxPrintout)`

See **external documentation**.

`getAbort(This) -> boolean()`

Types:

`This = wxPrinter()`

See **external documentation**.

`getLastError() -> wx_enum() (see module wx)`

See **external documentation**.
Res = ?wxPRINTER_NO_ERROR | ?wxPRINTER_CANCELLED | ?wxPRINTER_ERROR

`getPrintDialogData(This) -> wxPrintDialogData() (see module wxPrintDialogData)`

Types:

`This = wxPrinter()`

See **external documentation**.

print(This, Parent, Printout) -> boolean()

Types:

This = wxPrinter()
Parent = wxWindow() (see module wxWindow)
Printout = wxPrintout() (see module wxPrintout)

Equivalent to *print(This, Parent, Printout, [])*.

print(This, Parent, Printout, Option::[Option]) -> boolean()

Types:

This = wxPrinter()
Parent = wxWindow() (see module wxWindow)
Printout = wxPrintout() (see module wxPrintout)
Option = {prompt, boolean()}

See **external documentation**.

printDialog(This, Parent) -> wxDC() (see module wxDC)

Types:

This = wxPrinter()
Parent = wxWindow() (see module wxWindow)

See **external documentation**.

reportError(This, Parent, Printout, Message) -> ok

Types:

This = wxPrinter()
Parent = wxWindow() (see module wxWindow)
Printout = wxPrintout() (see module wxPrintout)
Message = chardata() (see module unicode)

See **external documentation**.

setup(This, Parent) -> boolean()

Types:

This = wxPrinter()
Parent = wxWindow() (see module wxWindow)

See **external documentation**.

destroy(This::wxPrinter()) -> ok

Destroys this object, do not use object again

wxPrintout

Erlang module

See external documentation: **wxPrintout**.

DATA TYPES

`wxPrintout()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Title::string(), OnPrintPage::function()) -> wxPrintout() (see module wxPrintout)`

@equiv new(Title, OnPrintPage, [])

`new(Title::string(), OnPrintPage::function(), Opts::[Option]) -> wxPrintout() (see module wxPrintout)`

Types:

```
Option = {onPreparePrinting, OnPreparePrinting::function()}
| {onBeginPrinting, OnBeginPrinting::function()} |
{onEndPrinting, OnEndPrinting::function()} | {onBeginDocument,
OnBeginDocument::function()} | {onEndDocument, OnEndDocument::function()}
| {hasPage, HasPage::function()} | {getPageInfo, GetPageInfo::function()}
```

Creates a wxPrintout object with a callback fun and optionally other callback funs.

```
OnPrintPage(This,Page) -> boolean()
```

```
OnPreparePrinting(This) -> term()
```

```
OnBeginPrinting(This) -> term()
```

```
OnEndPrinting(This) -> term()
```

```
OnBeginDocument(This,StartPage,EndPage) -> boolean()
```

```
OnEndDocument(This) -> term()
```

```
HasPage(This,Page)} -> boolean()
```

```
GetPageInfo(This) -> {MinPage:.integer(), MaxPage::integer(), PageFrom::integer(), PageTo::integer()}
```

The *This* argument is the wxPrintout object reference to this object
NOTE: The callbacks may not call other processes.

getDC(This) -> wxDC() (see module wxDC)

Types:

This = wxPrintout()

See **external documentation**.

getPageSizeMM(This) -> {W::integer(), H::integer()}

Types:

This = wxPrintout()

See **external documentation**.

getPageSizePixels(This) -> {W::integer(), H::integer()}

Types:

This = wxPrintout()

See **external documentation**.

getPaperRectPixels(This) -> {X::integer(), Y::integer(), W::integer(), H::integer()}

Types:

This = wxPrintout()

See **external documentation**.

getPPIPrinter(This) -> {X::integer(), Y::integer()}

Types:

This = wxPrintout()

See **external documentation**.

getPPIScreen(This) -> {X::integer(), Y::integer()}

Types:

This = wxPrintout()

See **external documentation**.

getTitle(This) -> charlist() (see module unicode)

Types:

This = wxPrintout()

See **external documentation**.

isPreview(This) -> boolean()

Types:

This = wxPrintout()

See **external documentation**.

fitThisSizeToPaper(This, ImageSize) -> ok

Types:

This = wxPrintout()
ImageSize = {W::integer(), H::integer()}

See **external documentation**.

fitThisSizeToPage(This, ImageSize) -> ok

Types:

This = wxPrintout()
ImageSize = {W::integer(), H::integer()}

See **external documentation**.

fitThisSizeToPageMargins(This, ImageSize, PageSetupData) -> ok

Types:

This = wxPrintout()
ImageSize = {W::integer(), H::integer()}
PageSetupData = wxPageSetupDialogData() (see module wxPageSetupDialogData)

See **external documentation**.

mapScreenSizeToPaper(This) -> ok

Types:

This = wxPrintout()

See **external documentation**.

mapScreenSizeToPage(This) -> ok

Types:

This = wxPrintout()

See **external documentation**.

mapScreenSizeToPageMargins(This, PageSetupData) -> ok

Types:

This = wxPrintout()
PageSetupData = wxPageSetupDialogData() (see module wxPageSetupDialogData)

See **external documentation**.

mapScreenSizeToDevice(This) -> ok

Types:

This = wxPrintout()

See **external documentation**.

getLogicalPaperRect(This) -> {X::integer(), Y::integer(), W::integer(), H::integer()}

Types:

```
This = wxPrintout()
```

See **external documentation**.

```
getLogicalPageRect(This) -> {X::integer(), Y::integer(), W::integer(),
H::integer()}
```

Types:

```
This = wxPrintout()
```

See **external documentation**.

```
getLogicalPageMarginsRect(This, PageSetupData) -> {X::integer(),
Y::integer(), W::integer(), H::integer()}
```

Types:

```
This = wxPrintout()
PageSetupData = wxPageSetupDialogData() (see module wxPageSetupDialogData)
```

See **external documentation**.

```
setLogicalOrigin(This, X, Y) -> ok
```

Types:

```
This = wxPrintout()
X = integer()
Y = integer()
```

See **external documentation**.

```
offsetLogicalOrigin(This, Xoff, Yoff) -> ok
```

Types:

```
This = wxPrintout()
Xoff = integer()
Yoff = integer()
```

See **external documentation**.

```
destroy(This::wxPrintout()) -> ok
```

Destroys this object, do not use object again

wxProgressDialog

Erlang module

See external documentation: **wxProgressDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxProgressDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Title, Message) -> wxProgressDialog()`

Types:

`Title = chardata() (see module unicode)`
`Message = chardata() (see module unicode)`

Equivalent to *new(Title, Message, [])*.

`new(Title, Message, Option::[Option]) -> wxProgressDialog()`

Types:

`Title = chardata() (see module unicode)`
`Message = chardata() (see module unicode)`
`Option = {maximum, integer()} | {parent, wxWindow() (see module wxWindow)} | {style, integer()}`

See **external documentation**.

`resume(This) -> ok`

Types:

`This = wxProgressDialog()`

See **external documentation**.

`update(This) -> ok`

Types:

`This = wxProgressDialog()`

See **external documentation**.

`update(This, Value) -> boolean()`

Types:

```
This = wxProgressDialog()
Value = integer()
```

Equivalent to *update(This, Value, [])*.

update(This, Value, Option::[Option]) -> boolean()

Types:

```
This = wxProgressDialog()
Value = integer()
Option = {newmsg, chardata() (see module unicode)}
```

See **external documentation**.

destroy(This::wxProgressDialog()) -> ok

Destroys this object, do not use object again

wxQueryNewPaletteEvent

Erlang module

See external documentation: **wxQueryNewPaletteEvent**.

Use *wxEvtHandler:connect/3* with EventType:

query_new_palette

See also the message variant *#wxQueryNewPalette{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxQueryNewPaletteEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`setPaletteRealized(This, Realized) -> ok`

Types:

```
This = wxQueryNewPaletteEvent()
Realized = boolean()
```

See **external documentation**.

`getPaletteRealized(This) -> boolean()`

Types:

```
This = wxQueryNewPaletteEvent()
```

See **external documentation**.

wxRadioBox

Erlang module

See external documentation: **wxRadioBox**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxRadioBox()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent, Id, Title, Pos, Size, Choices) -> wxRadioBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
```

Equivalent to *new(Parent, Id, Title, Pos, Size, Choices, [])*.

`new(Parent, Id, Title, Pos, Size, Choices, Option::[Option]) -> wxRadioBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
Option = {majorDim, integer()} | {style, integer()} | {val, wx_object()
(see module wx)}
```

See **external documentation**.

`create(This, Parent, Id, Title, Pos, Size, Choices) -> boolean()`

Types:

```
This = wxRadioBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

```
Title = chardata() (see module unicode)
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
```

Equivalent to *create(This, Parent, Id, Title, Pos, Size, Choices, [])*.

create(This, Parent, Id, Title, Pos, Size, Choices, Option::[Option]) -> boolean()

Types:

```
This = wxRadioBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Title = chardata() (see module unicode)
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}
Choices = [chardata() (see module unicode)]
Option = {majorDim, integer()} | {style, integer()} | {val, wx_object()
(see module wx)}
```

See **external documentation**.

enable(This) -> boolean()

Types:

```
This = wxRadioBox()
```

Equivalent to *enable(This, [])*.

enable(This, N) -> boolean()

Types:

```
This = wxRadioBox()
N = integer()
```

See **external documentation**.
Also:
enable(This, [Option]) -> boolean() when
This::wxRadioBox(),
Option :: {enable, boolean()}.

enable(This, N, Option::[Option]) -> boolean()

Types:

```
This = wxRadioBox()
N = integer()
Option = {enable, boolean()}
```

See **external documentation**.

getSelection(This) -> integer()

Types:

```
This = wxRadioBox()
```

See **external documentation**.

getString(This, N) -> charlist() (see module unicode)

Types:

```
This = wxRadioBox()
N = integer()
```

See **external documentation**.

setSelection(This, N) -> ok

Types:

```
This = wxRadioBox()
N = integer()
```

See **external documentation**.

show(This) -> boolean()

Types:

```
This = wxRadioBox()
```

Equivalent to *show(This, [])*.

show(This, N) -> boolean()

Types:

```
This = wxRadioBox()
N = integer()
```

See **external documentation**.
Also:
show(This, [Option]) -> boolean() when
This::wxRadioBox(),
Option :: {show, boolean()}.

show(This, N, Option::[Option]) -> boolean()

Types:

```
This = wxRadioBox()
N = integer()
Option = {show, boolean()}
```

See **external documentation**.

getColumnCount(This) -> integer()

Types:

```
This = wxRadioBox()
```

See **external documentation**.

getItemHelpText(This, N) -> charlist() (see module unicode)

Types:

```
This = wxRadioBox()
N = integer()
```

See **external documentation**.

getItemToolTip(This, Item) -> wxToolTip() (see module wxToolTip)

Types:

```
This = wxRadioBox()
Item = integer()
```

See **external documentation**.

getItemFromPoint(This, Pt) -> integer()

Types:

```
This = wxRadioBox()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

getRowCount(This) -> integer()

Types:

```
This = wxRadioBox()
```

See **external documentation**.

isItemEnabled(This, N) -> boolean()

Types:

```
This = wxRadioBox()
N = integer()
```

See **external documentation**.

isItemShown(This, N) -> boolean()

Types:

```
This = wxRadioBox()
N = integer()
```

See **external documentation**.

setItemHelpText(This, N, HelpText) -> ok

Types:

```
This = wxRadioBox()
N = integer()
HelpText = chardata() (see module unicode)
```

See **external documentation**.

setItemToolTip(This, Item, Text) -> ok

Types:

```
This = wxRadioBox()
```

```
Item = integer()
Text = chardata() (see module unicode)
```

See **external documentation**.

destroy(This::wxRadioBox()) -> ok

Destroys this object, do not use object again

wxRadioButton

Erlang module

See external documentation: **wxRadioButton**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxRadioButton()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxRadioButton()`

See **external documentation**.

`new(Parent, Id, Label) -> wxRadioButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Label, [])*.

`new(Parent, Id, Label, Option::[Option]) -> wxRadioButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

`create(This, Parent, Id, Label) -> boolean()`

Types:

```
This = wxRadioButton()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Label, [])*.

create(This, Parent, Id, Label, Option::[Option]) -> boolean()

Types:

This = wxRadioButton()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Label = chardata() (see module unicode)

Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {validator, wx_object() (see module wx)}

See **external documentation**.

getValue(This) -> boolean()

Types:

This = wxRadioButton()

See **external documentation**.

setValue(This, Val) -> ok

Types:

This = wxRadioButton()

Val = boolean()

See **external documentation**.

destroy(This::wxRadioButton()) -> ok

Destroys this object, do not use object again

wxRegion

Erlang module

See external documentation: **wxRegion**.

DATA TYPES

`wxRegion()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxRegion()`

See **external documentation**.

`new(Bmp) -> wxRegion()`

Types:

`Bmp = wxBitmap() (see module wxBitmap)`

See **external documentation**.
Also:
new(Rect) -> wxRegion() when
Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

`new(TopLeft, BottomRight) -> wxRegion()`

Types:

`TopLeft = {X::integer(), Y::integer()}`
`BottomRight = {X::integer(), Y::integer()}`

See **external documentation**.

`new(X, Y, W, H) -> wxRegion()`

Types:

`X = integer()`
`Y = integer()`
`W = integer()`
`H = integer()`

See **external documentation**.

`clear(This) -> ok`

Types:

`This = wxRegion()`

See **external documentation**.

contains(This, Pt) -> wx_enum() (see module wx)

Types:

This = wxRegion()

Pt = {X::integer(), Y::integer()}

See **external documentation**.

Also:

contains(This, Rect) -> wx:wx_enum() when

This::wxRegion(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

Res = ?wxOutRegion | ?wxPartRegion | ?wxInRegion

contains(This, X, Y) -> wx_enum() (see module wx)

Types:

This = wxRegion()

X = integer()

Y = integer()

See **external documentation**.

Res = ?wxOutRegion | ?wxPartRegion | ?wxInRegion

contains(This, X, Y, W, H) -> wx_enum() (see module wx)

Types:

This = wxRegion()

X = integer()

Y = integer()

W = integer()

H = integer()

See **external documentation**.

Res = ?wxOutRegion | ?wxPartRegion | ?wxInRegion

convertToBitmap(This) -> wxBitmap() (see module wxBitmap)

Types:

This = wxRegion()

See **external documentation**.

getBox(This) -> {X::integer(), Y::integer(), W::integer(), H::integer()}

Types:

This = wxRegion()

See **external documentation**.

intersect(This, Region) -> boolean()

Types:

This = wxRegion()

Region = wxRegion()

See **external documentation**.

Also:

intersect(This, Rect) -> boolean() when
This::wxRegion(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

`intersect(This, X, Y, W, H) -> boolean()`

Types:

```
This = wxRegion()
X = integer()
Y = integer()
W = integer()
H = integer()
```

See **external documentation**.

`isEmpty(This) -> boolean()`

Types:

```
This = wxRegion()
```

See **external documentation**.

`subtract(This, Region) -> boolean()`

Types:

```
This = wxRegion()
Region = wxRegion()
```

See **external documentation**.
Also:
subtract(This, Rect) -> boolean() when
This::wxRegion(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

`subtract(This, X, Y, W, H) -> boolean()`

Types:

```
This = wxRegion()
X = integer()
Y = integer()
W = integer()
H = integer()
```

See **external documentation**.

`offset(This, Pt) -> boolean()`

Types:

```
This = wxRegion()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

`offset(This, X, Y) -> boolean()`

Types:

```
This = wxRegion()
```

```
X = integer()
Y = integer()
```

See **external documentation**.

union(This, Region) -> boolean()

Types:

```
This = wxRegion()
Region = wxRegion() | wxBitmap() (see module wxBitmap)
```

See **external documentation**.
Also:
union(This, Rect) -> boolean() when
This::wxRegion(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

union(This, Bmp, Transp) -> boolean()

Types:

```
This = wxRegion()
Bmp = wxBitmap() (see module wxBitmap)
Transp = wx_colour() (see module wx)
```

Equivalent to *union(This, Bmp, Transp, [])*.

union(This, Bmp, Transp, Option::[Option]) -> boolean()

Types:

```
This = wxRegion()
Bmp = wxBitmap() (see module wxBitmap)
Transp = wx_colour() (see module wx)
Option = {tolerance, integer()}
```

See **external documentation**.

union(This, X, Y, W, H) -> boolean()

Types:

```
This = wxRegion()
X = integer()
Y = integer()
W = integer()
H = integer()
```

See **external documentation**.

Xor(This, Region) -> boolean()

Types:

```
This = wxRegion()
Region = wxRegion()
```

See **external documentation**.
Also:
'Xor'(This, Rect) -> boolean() when

This::wxRegion(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

Xor(This, X, Y, W, H) -> boolean()

Types:

```
This = wxRegion()
X = integer()
Y = integer()
W = integer()
H = integer()
```

See **external documentation**.

destroy(This::wxRegion()) -> ok

Destroys this object, do not use object again

wxSashEvent

Erlang module

See external documentation: **wxSashEvent**.

Use *wxEvtHandler:connect/3* with EventType:

sash_dragged

See also the message variant *#wxSash{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxSashEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getEdge(This) -> wx_enum() (see module wx)`

Types:

`This = wxSashEvent()`

See **external documentation**.
Res = ?wxSASH_TOP | ?wxSASH_RIGHT | ?wxSASH_BOTTOM | ?wxSASH_LEFT | ?wxSASH_NONE

`getDragRect(This) -> {X::integer(), Y::integer(), W::integer(), H::integer()}`

Types:

`This = wxSashEvent()`

See **external documentation**.

`getDragStatus(This) -> wx_enum() (see module wx)`

Types:

`This = wxSashEvent()`

See **external documentation**.
Res = ?wxSASH_STATUS_OK | ?wxSASH_STATUS_OUT_OF_RANGE

wxSashLayoutWindow

Erlang module

See external documentation: **wxSashLayoutWindow**.

This class is derived (and can use functions) from:
wxSashWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxSashLayoutWindow()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSashLayoutWindow()`

See **external documentation**.

`new(Parent) -> wxSashLayoutWindow()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxSashLayoutWindow()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

```
This = wxSashLayoutWindow()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

```
This = wxSashLayoutWindow()
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

getAlignment(This) -> wx_enum() (see module wx)

Types:

This = wxSashLayoutWindow()

See **external documentation**.
Res = ?wxLAYOUT_NONE | ?wxLAYOUT_TOP | ?wxLAYOUT_LEFT | ?wxLAYOUT_RIGHT | ?wxLAYOUT_BOTTOM

getOrientation(This) -> wx_enum() (see module wx)

Types:

This = wxSashLayoutWindow()

See **external documentation**.
Res = ?wxLAYOUT_HORIZONTAL | ?wxLAYOUT_VERTICAL

setAlignment(This, Align) -> ok

Types:

This = wxSashLayoutWindow()
Align = wx_enum() (see module wx)

See **external documentation**.
Align = ?wxLAYOUT_NONE | ?wxLAYOUT_TOP | ?wxLAYOUT_LEFT | ?wxLAYOUT_RIGHT | ?wxLAYOUT_BOTTOM

setDefaultSize(This, Size) -> ok

Types:

This = wxSashLayoutWindow()
Size = {W::integer(), H::integer()}

See **external documentation**.

setOrientation(This, Orient) -> ok

Types:

This = wxSashLayoutWindow()
Orient = wx_enum() (see module wx)

See **external documentation**.
Orient = ?wxLAYOUT_HORIZONTAL | ?wxLAYOUT_VERTICAL

destroy(This::wxSashLayoutWindow()) -> ok

Destroys this object, do not use object again

wxSashWindow

Erlang module

See external documentation: **wxSashWindow**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxSashWindow()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSashWindow()`

See **external documentation**.

`new(Parent) -> wxSashWindow()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxSashWindow()`

Types:

`Parent = wxWindow() (see module wxWindow)`

`Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}`

See **external documentation**.

`getSashVisible(This, Edge) -> boolean()`

Types:

`This = wxSashWindow()`

`Edge = wx_enum() (see module wx)`

See **external documentation**.
Edge = ?wxSASH_TOP | ?wxSASH_RIGHT | ?wxSASH_BOTTOM | ?wxSASH_LEFT | ?wxSASH_NONE

`getMaximumSizeX(This) -> integer()`

Types:

`This = wxSashWindow()`

See **external documentation**.

getMaximumSizeY(This) -> integer()

Types:

This = wxSashWindow()

See **external documentation**.

getMinimumSizeX(This) -> integer()

Types:

This = wxSashWindow()

See **external documentation**.

getMinimumSizeY(This) -> integer()

Types:

This = wxSashWindow()

See **external documentation**.

setMaximumSizeX(This, Max) -> ok

Types:

This = wxSashWindow()

Max = integer()

See **external documentation**.

setMaximumSizeY(This, Max) -> ok

Types:

This = wxSashWindow()

Max = integer()

See **external documentation**.

setMinimumSizeX(This, Min) -> ok

Types:

This = wxSashWindow()

Min = integer()

See **external documentation**.

setMinimumSizeY(This, Min) -> ok

Types:

This = wxSashWindow()

Min = integer()

See **external documentation**.

setSashVisible(This, Edge, Sash) -> ok

Types:

This = wxSashWindow()

Edge = wx_enum() (see module wx)

Sash = boolean()

See **external documentation**.
Edge = ?wxSASH_TOP | ?wxSASH_RIGHT | ?wxSASH_BOTTOM | ?wxSASH_LEFT | ?wxSASH_NONE

destroy(This::wxSashWindow()) -> ok

Destroys this object, do not use object again

wxScreenDC

Erlang module

See external documentation: **wxScreenDC**.

This class is derived (and can use functions) from:
wxDC

DATA TYPES

`wxScreenDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxScreenDC()`

See **external documentation**.

`destroy(This::wxScreenDC()) -> ok`

Destroys this object, do not use object again

wxScrollBar

Erlang module

See external documentation: **wxScrollBar**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxScrollBar()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxScrollBar()`

See **external documentation**.

`new(Parent, Id) -> wxScrollBar()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxScrollBar()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

`create(This, Parent, Id) -> boolean()`

Types:

```
This = wxScrollBar()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

`create(This, Parent, Id, Option::[Option]) -> boolean()`

Types:

```
This = wxScrollBar()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

getRange(This) -> integer()

Types:

```
This = wxScrollBar()
```

See **external documentation**.

getPageSize(This) -> integer()

Types:

```
This = wxScrollBar()
```

See **external documentation**.

getThumbPosition(This) -> integer()

Types:

```
This = wxScrollBar()
```

See **external documentation**.

getThumbSize(This) -> integer()

Types:

```
This = wxScrollBar()
```

See **external documentation**.

setThumbPosition(This, ViewStart) -> ok

Types:

```
This = wxScrollBar()
ViewStart = integer()
```

See **external documentation**.

setScrollbar(This, Position, ThumbSize, Range, PageSize) -> ok

Types:

```
This = wxScrollBar()
Position = integer()
ThumbSize = integer()
Range = integer()
PageSize = integer()
```

Equivalent to *setScrollbar(This, Position, ThumbSize, Range, PageSize, [])*.

setScrollbar(This, Position, ThumbSize, Range, PageSize, Option::[Option]) -> ok

Types:

```
This = wxScrollBar()
Position = integer()
ThumbSize = integer()
Range = integer()
PageSize = integer()
Option = {refresh, boolean()}
```

See **external documentation**.

destroy(This::wxScrollBar()) -> ok

Destroys this object, do not use object again

wxScrollEvent

Erlang module

See external documentation: **wxScrollEvent**.

Use *wxEvtHandler:connect/3* with EventType:

scroll_top, *scroll_bottom*, *scroll_lineup*, *scroll_linedown*, *scroll_pageup*, *scroll_pagedown*, *scroll_thumbtrack*, *scroll_thumbrelease*, *scroll_changed*

See also the message variant *#wxScroll{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxScrollEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getOrientation(This) -> integer()`

Types:

`This = wxScrollEvent()`

See **external documentation**.

`getPosition(This) -> integer()`

Types:

`This = wxScrollEvent()`

See **external documentation**.

wxScrollWinEvent

Erlang module

See external documentation: **wxScrollWinEvent**.

Use *wxEvtHandler:connect/3* with EventType:

scrollwin_top, *scrollwin_bottom*, *scrollwin_lineup*, *scrollwin_linedown*, *scrollwin_pageup*, *scrollwin_pagedown*, *scrollwin_thumbtrack*, *scrollwin_thumbrelease*

See also the message variant *#wxScrollWin{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxScrollWinEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getOrientation(This) -> integer()`

Types:

`This = wxScrollWinEvent()`

See **external documentation**.

`getPosition(This) -> integer()`

Types:

`This = wxScrollWinEvent()`

See **external documentation**.

wxScrolledWindow

Erlang module

See external documentation: **wxScrolledWindow**.

This class is derived (and can use functions) from:
wxPanel
wxWindow
wxEvtHandler

DATA TYPES

`wxScrolledWindow()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxScrolledWindow()`

See **external documentation**.

`new(Parent) -> wxScrolledWindow()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxScrolledWindow()`

Types:

`Parent = wxWindow() (see module wxWindow)`

`Option = {winid, integer()} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}`

See **external documentation**.

`calcScrolledPosition(This, Pt) -> {X::integer(), Y::integer()}`

Types:

`This = wxScrolledWindow()`

`Pt = {X::integer(), Y::integer()}`

See **external documentation**.

`calcScrolledPosition(This, X, Y) -> {Xx::integer(), Yy::integer()}`

Types:

`This = wxScrolledWindow()`

`X = integer()`

`Y = integer()`

See **external documentation**.

calcUnscrolledPosition(This, Pt) -> {X::integer(), Y::integer()}

Types:

This = wxScrolledWindow()

Pt = {X::integer(), Y::integer()}

See **external documentation**.

calcUnscrolledPosition(This, X, Y) -> {Xx::integer(), Yy::integer()}

Types:

This = wxScrolledWindow()

X = integer()

Y = integer()

See **external documentation**.

enableScrolling(This, X_scrolling, Y_scrolling) -> ok

Types:

This = wxScrolledWindow()

X_scrolling = boolean()

Y_scrolling = boolean()

See **external documentation**.

getScrollPixelsPerUnit(This) -> {PixelsPerUnitX::integer(), PixelsPerUnitY::integer()}

Types:

This = wxScrolledWindow()

See **external documentation**.

getViewStart(This) -> {X::integer(), Y::integer()}

Types:

This = wxScrolledWindow()

See **external documentation**.

doPrepareDC(This, Dc) -> ok

Types:

This = wxScrolledWindow()

Dc = wxDC() (see module wxDC)

See **external documentation**.

prepareDC(This, Dc) -> ok

Types:

This = wxScrolledWindow()

Dc = wxDC() (see module wxDC)

See **external documentation**.

scroll(This, X, Y) -> ok

Types:

This = wxScrolledWindow()
X = integer()
Y = integer()

See **external documentation**.

setScrollbars(This, PixelsPerUnitX, PixelsPerUnitY, NoUnitsX, NoUnitsY) -> ok

Types:

This = wxScrolledWindow()
PixelsPerUnitX = integer()
PixelsPerUnitY = integer()
NoUnitsX = integer()
NoUnitsY = integer()

Equivalent to *setScrollbars(This, PixelsPerUnitX, PixelsPerUnitY, NoUnitsX, NoUnitsY, [])*.

setScrollbars(This, PixelsPerUnitX, PixelsPerUnitY, NoUnitsX, NoUnitsY, Option::[Option]) -> ok

Types:

This = wxScrolledWindow()
PixelsPerUnitX = integer()
PixelsPerUnitY = integer()
NoUnitsX = integer()
NoUnitsY = integer()
Option = {xPos, integer()} | {yPos, integer()} | {noRefresh, boolean()}

See **external documentation**.

setScrollRate(This, Xstep, Ystep) -> ok

Types:

This = wxScrolledWindow()
Xstep = integer()
Ystep = integer()

See **external documentation**.

setTargetWindow(This, Target) -> ok

Types:

This = wxScrolledWindow()
Target = wxWindow() (see module wxWindow)

See **external documentation**.

destroy(This::wxScrolledWindow()) -> ok

Destroys this object, do not use object again

wxSetCursorEvent

Erlang module

See external documentation: **wxSetCursorEvent**.

Use *wxEvtHandler:connect/3* with EventType:

set_cursor

See also the message variant *#wxSetCursor{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxSetCursorEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getCursor(This) -> wxCursor() (see module wxCursor)`

Types:

`This = wxSetCursorEvent()`

See **external documentation**.

`getX(This) -> integer()`

Types:

`This = wxSetCursorEvent()`

See **external documentation**.

`getY(This) -> integer()`

Types:

`This = wxSetCursorEvent()`

See **external documentation**.

`hasCursor(This) -> boolean()`

Types:

`This = wxSetCursorEvent()`

See **external documentation**.

`setCursor(This, Cursor) -> ok`

Types:

`This = wxSetCursorEvent()`
`Cursor = wxCursor() (see module wxCursor)`

See **external documentation**.

wxShowEvent

Erlang module

See external documentation: **wxShowEvent**.

Use *wxEvtHandler:connect/3* with EventType:

show

See also the message variant *#wxShow{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxShowEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`setShow(This, Show) -> ok`

Types:

`This = wxShowEvent()`
`Show = boolean()`

See **external documentation**.

`getShow(This) -> boolean()`

Types:

`This = wxShowEvent()`

See **external documentation**.

wxSingleChoiceDialog

Erlang module

See external documentation: **wxSingleChoiceDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxSingleChoiceDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSingleChoiceDialog()`

See **external documentation**.

`new(Parent, Message, Caption, Choices) -> wxSingleChoiceDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Message = chardata() (see module unicode)
Caption = chardata() (see module unicode)
Choices = [chardata() (see module unicode)]
```

Equivalent to *new(Parent, Message, Caption, Choices, [])*.

`new(Parent, Message, Caption, Choices, Option::[Option]) -> wxSingleChoiceDialog()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Message = chardata() (see module unicode)
Caption = chardata() (see module unicode)
Choices = [chardata() (see module unicode)]
Option = {style, integer()} | {pos, {X::integer(), Y::integer()}}
```

See **external documentation**.

`getSelection(This) -> integer()`

Types:

```
This = wxSingleChoiceDialog()
```

See **external documentation**.

getStringSelection(This) -> charlist() (see module unicode)

Types:

This = wxSingleChoiceDialog()

See **external documentation**.

setSelection(This, Sel) -> ok

Types:

This = wxSingleChoiceDialog()

Sel = integer()

See **external documentation**.

destroy(This::wxSingleChoiceDialog()) -> ok

Destroys this object, do not use object again

wxSizeEvent

Erlang module

See external documentation: **wxSizeEvent**.

Use *wxEvtHandler:connect/3* with EventType:

size

See also the message variant *#wxSize{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxSizeEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getSize(This) -> {W::integer(), H::integer()}`

Types:

`This = wxSizeEvent()`

See **external documentation**.

wxSizer

Erlang module

See external documentation: **wxSizer**.

DATA TYPES

`wxSizer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`add(This, Window) -> wxSizerItem() (see module wxSizerItem)`

Types:

`This = wxSizer()`
`Window = wxWindow() (see module wxWindow) | wxSizer()`

Equivalent to *add(This, Window, [])*.

`add(This, Width, Height) -> wxSizerItem() (see module wxSizerItem)`

Types:

`This = wxSizer()`
`Width = integer()`
`Height = integer()`

See **external documentation**.
Also:
add(This, Window, [Option]) -> wxSizerItem:wxSizerItem() when
This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer(),
Option :: {proportion, integer()}
| {flag, integer()}
| {border, integer()}
| {userData, wx:wx_object()};
(This, Window, Flags) -> wxSizerItem:wxSizerItem() when
This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer(), Flags::wxSizerFlags:wxSizerFlags().

`add(This, Width, Height, Option::[Option]) -> wxSizerItem() (see module wxSizerItem)`

Types:

`This = wxSizer()`
`Width = integer()`
`Height = integer()`
`Option = {proportion, integer()} | {flag, integer()} | {border, integer()} | {userData, wx_object() (see module wx)}`

See **external documentation**.

addSpacer(This, Size) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Size = integer()

See **external documentation**.

addStretchSpacer(This) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Equivalent to *addStretchSpacer(This, [])*.

addStretchSpacer(This, Option::[Option]) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Option = {prop, integer()}

See **external documentation**.

calcMin(This) -> {W::integer(), H::integer()}

Types:

This = wxSizer()

See **external documentation**.

clear(This) -> ok

Types:

This = wxSizer()

Equivalent to *clear(This, [])*.

clear(This, Option::[Option]) -> ok

Types:

This = wxSizer()

Option = {delete_windows, boolean()}

See **external documentation**.

detach(This, Index) -> boolean()

Types:

This = wxSizer()

Index = integer()

See **external documentation**.

Also:

detach(This, Window) -> boolean() when

This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer().

fit(This, Window) -> {W::integer(), H::integer()}

Types:

This = wxSizer()

Window = wxWindow() (see module wxWindow)

See **external documentation**.

fitInside(This, Window) -> ok

Types:

This = wxSizer()

Window = wxWindow() (see module wxWindow)

See **external documentation**.

getChildren(This) -> [wxSizerItem() (see module wxSizerItem)]

Types:

This = wxSizer()

See **external documentation**.

getItem(This, Window) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Window = wxWindow() (see module wxWindow) | wxSizer()

See **external documentation**.

Also:

getItem(This, Index) -> wxSizerItem:wxSizerItem() when This::wxSizer(), Index::integer().

getItem(This, Window, Option::[Option]) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Window = wxWindow() (see module wxWindow) | wxSizer()

Option = {recursive, boolean()}

See **external documentation**.

getSize(This) -> {W::integer(), H::integer()}

Types:

This = wxSizer()

See **external documentation**.

getPosition(This) -> {X::integer(), Y::integer()}

Types:

This = wxSizer()

See **external documentation**.

getMinSize(This) -> {W::integer(), H::integer()}

Types:

This = wxSizer()

See **external documentation**.

hide(This, Window) -> boolean()

Types:

This = wxSizer()

Window = wxWindow() (see module wxWindow) | wxSizer()

See **external documentation**.

Also:

hide(This, Index) -> boolean() when

This::wxSizer(), Index::integer().

hide(This, Window, Option::[Option]) -> boolean()

Types:

This = wxSizer()

Window = wxWindow() (see module wxWindow) | wxSizer()

Option = {recursive, boolean()}

See **external documentation**.

insert(This, Index, Item) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Index = integer()

Item = wxSizerItem() (see module wxSizerItem)

See **external documentation**.

insert(This, Index, Width, Height) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Index = integer()

Width = integer()

Height = integer()

See **external documentation**.

Also:

insert(This, Index, Window, [Option]) -> wxSizerItem:wxSizerItem() when

This::wxSizer(), Index::integer(), Window::wxWindow:wxWindow() | wxSizer(),

Option :: {proportion, integer()}

| {flag, integer()}

| {border, integer()}

| {userData, wx:wx_object()};

(This, Index, Window, Flags) -> wxSizerItem:wxSizerItem() when

This::wxSizer(), Index::integer(), Window::wxWindow:wxWindow() | wxSizer(),

Flags::wxSizerFlags:wxSizerFlags().

insert(This, Index, Width, Height, Option::[Option]) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()
Index = integer()
Width = integer()
Height = integer()
Option = {proportion, integer()} | {flag, integer()} | {border, integer()} | {userData, wx_object() (see module wx)}

See **external documentation**.

insertSpacer(This, Index, Size) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()
Index = integer()
Size = integer()

See **external documentation**.

insertStretchSpacer(This, Index) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()
Index = integer()

Equivalent to *insertStretchSpacer(This, Index, [])*.

insertStretchSpacer(This, Index, Option::[Option]) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()
Index = integer()
Option = {prop, integer()}

See **external documentation**.

isShown(This, Index) -> boolean()

Types:

This = wxSizer()
Index = integer()

See **external documentation**.
Also:
isShown(This, Window) -> boolean() when
This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer().

layout(This) -> ok

Types:

This = wxSizer()

See **external documentation**.

prepend(This, Item) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Item = wxSizerItem() (see module wxSizerItem)

See **external documentation**.

prepend(This, Width, Height) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Width = integer()

Height = integer()

See **external documentation**.

Also:

prepend(This, Window, [Option]) -> wxSizerItem:wxSizerItem() when

This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer(),

Option :: {proportion, integer()}

| {flag, integer()}

| {border, integer()}

| {userData, wx:wx_object()};

(This, Window, Flags) -> wxSizerItem:wxSizerItem() when

This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer(), Flags::wxSizerFlags:wxSizerFlags().

prepend(This, Width, Height, Option::[Option]) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Width = integer()

Height = integer()

Option = {proportion, integer()} | {flag, integer()} | {border, integer()} | {userData, wx_object() (see module wx)}

See **external documentation**.

prependSpacer(This, Size) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Size = integer()

See **external documentation**.

prependStretchSpacer(This) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Equivalent to *prependStretchSpacer(This, [])*.

prependStretchSpacer(This, Option::[Option]) -> wxSizerItem() (see module wxSizerItem)

Types:

This = wxSizer()

Option = {prop, integer()}

See **external documentation**.

recalcSizes(This) -> ok

Types:

This = wxSizer()

See **external documentation**.

remove(This, Index) -> boolean()

Types:

This = wxSizer()

Index = integer()

See **external documentation**.
Also:
remove(This, Sizer) -> boolean() when
This::wxSizer(), Sizer::wxSizer().

replace(This, Oldwin, Newwin) -> boolean()

Types:

This = wxSizer()

Oldwin = wxWindow() (see module wxWindow) | wxSizer()

Newwin = wxWindow() (see module wxWindow) | wxSizer()

See **external documentation**.
Also:
replace(This, Index, Newitem) -> boolean() when
This::wxSizer(), Index::integer(), Newitem::wxSizerItem:wxSizerItem().

replace(This, Oldwin, Newwin, Option::[Option]) -> boolean()

Types:

This = wxSizer()

Oldwin = wxWindow() (see module wxWindow) | wxSizer()

Newwin = wxWindow() (see module wxWindow) | wxSizer()

Option = {recursive, boolean()}

See **external documentation**.

setDimension(This, X, Y, Width, Height) -> ok

Types:

This = wxSizer()

X = integer()

Y = integer()

```
Width = integer()
Height = integer()
```

See **external documentation**.

setMinSize(This, Size) -> ok

Types:

```
This = wxSizer()
Size = {W::integer(), H::integer()}
```

See **external documentation**.

setMinSize(This, Width, Height) -> ok

Types:

```
This = wxSizer()
Width = integer()
Height = integer()
```

See **external documentation**.

setItemMinSize(This, Index, Size) -> boolean()

Types:

```
This = wxSizer()
Index = integer()
Size = {W::integer(), H::integer()}
```

See **external documentation**.
Also:
setItemMinSize(This, Window, Size) -> boolean() when
This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer(), Size::{W::integer(), H::integer()}.

setItemMinSize(This, Index, Width, Height) -> boolean()

Types:

```
This = wxSizer()
Index = integer()
Width = integer()
Height = integer()
```

See **external documentation**.
Also:
setItemMinSize(This, Window, Width, Height) -> boolean() when
This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer(), Width::integer(), Height::integer().

setSizeHints(This, Window) -> ok

Types:

```
This = wxSizer()
Window = wxWindow() (see module wxWindow)
```

See **external documentation**.

setVirtualSizeHints(This, Window) -> ok

Types:

This = wxSizer()

Window = wxWindow() (see module wxWindow)

See **external documentation**.

show(This, Index) -> boolean()

Types:

This = wxSizer()

Index = integer()

See **external documentation**.
Also:
show(This, Window) -> boolean() when
This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer();
(This, Show) -> ok when
This::wxSizer(), Show::boolean().

show(This, Index, Option::[Option]) -> boolean()

Types:

This = wxSizer()

Index = integer()

Option = {show, boolean()}

See **external documentation**.
Also:
show(This, Window, [Option]) -> boolean() when
This::wxSizer(), Window::wxWindow:wxWindow() | wxSizer(),
Option :: {show, boolean()}
| {recursive, boolean()}.

wxSizerFlags

Erlang module

See external documentation: **wxSizerFlags**.

DATA TYPES

`wxSizerFlags()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSizerFlags()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxSizerFlags()`

Types:

```
Option = {proportion, integer()}
```

See **external documentation**.

`align(This, Alignment) -> wxSizerFlags()`

Types:

```
This = wxSizerFlags()
Alignment = integer()
```

See **external documentation**.

`border(This) -> wxSizerFlags()`

Types:

```
This = wxSizerFlags()
```

Equivalent to *border(This, [])*.

`border(This, Option::[Option]) -> wxSizerFlags()`

Types:

```
This = wxSizerFlags()
Option = {direction, integer()}
```

See **external documentation**.

`border(This, Direction, BorderInPixels) -> wxSizerFlags()`

Types:

```
This = wxSizerFlags()
Direction = integer()
BorderInPixels = integer()
```

See **external documentation**.

center(This) -> wxSizerFlags()

Types:

This = wxSizerFlags()

See **external documentation**.

centre(This) -> wxSizerFlags()

Types:

This = wxSizerFlags()

See **external documentation**.

expand(This) -> wxSizerFlags()

Types:

This = wxSizerFlags()

See **external documentation**.

left(This) -> wxSizerFlags()

Types:

This = wxSizerFlags()

See **external documentation**.

proportion(This, Proportion) -> wxSizerFlags()

Types:

This = wxSizerFlags()

Proportion = integer()

See **external documentation**.

right(This) -> wxSizerFlags()

Types:

This = wxSizerFlags()

See **external documentation**.

destroy(This::wxSizerFlags()) -> ok

Destroys this object, do not use object again

wxSizerItem

Erlang module

See external documentation: **wxSizerItem**.

DATA TYPES

`wxSizerItem()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSizerItem()`

See **external documentation**.

`new(Window, Flags) -> wxSizerItem()`

Types:

```
Window = wxWindow() (see module wxWindow) | wxSizer() (see module wxSizer)
Flags = wxSizerFlags() (see module wxSizerFlags)
```

See **external documentation**.

`new(Width, Height, Flags) -> wxSizerItem()`

Types:

```
Width = integer()
Height = integer()
Flags = wxSizerFlags() (see module wxSizerFlags)
```

See **external documentation**.

`new(Window, Proportion, Flag, Border, UserData) -> wxSizerItem()`

Types:

```
Window = wxWindow() (see module wxWindow) | wxSizer() (see module wxSizer)
Proportion = integer()
Flag = integer()
Border = integer()
UserData = wx_object() (see module wx)
```

See **external documentation**.

`new(Width, Height, Proportion, Flag, Border, UserData) -> wxSizerItem()`

Types:

```
Width = integer()
Height = integer()
Proportion = integer()
```

```
Flag = integer()
Border = integer()
UserData = wx_object() (see module wx)
```

See **external documentation**.

calcMin(This) -> {W::integer(), H::integer()}

Types:

```
This = wxSizerItem()
```

See **external documentation**.

deleteWindows(This) -> ok

Types:

```
This = wxSizerItem()
```

See **external documentation**.

detachSizer(This) -> ok

Types:

```
This = wxSizerItem()
```

See **external documentation**.

getBorder(This) -> integer()

Types:

```
This = wxSizerItem()
```

See **external documentation**.

getFlag(This) -> integer()

Types:

```
This = wxSizerItem()
```

See **external documentation**.

getMinSize(This) -> {W::integer(), H::integer()}

Types:

```
This = wxSizerItem()
```

See **external documentation**.

getPosition(This) -> {X::integer(), Y::integer()}

Types:

```
This = wxSizerItem()
```

See **external documentation**.

getProportion(This) -> integer()

Types:

```
This = wxSizerItem()
```

See **external documentation**.

getRatio(This) -> number()

Types:

This = wxSizerItem()

See **external documentation**.

getRect(This) -> {X::integer(), Y::integer(), W::integer(), H::integer()}

Types:

This = wxSizerItem()

See **external documentation**.

getSize(This) -> {W::integer(), H::integer()}

Types:

This = wxSizerItem()

See **external documentation**.

getSizer(This) -> wxSizer() (see module wxSizer)

Types:

This = wxSizerItem()

See **external documentation**.

getSpacer(This) -> {W::integer(), H::integer()}

Types:

This = wxSizerItem()

See **external documentation**.

getUserData(This) -> wx_object() (see module wx)

Types:

This = wxSizerItem()

See **external documentation**.

getWindow(This) -> wxWindow() (see module wxWindow)

Types:

This = wxSizerItem()

See **external documentation**.

isSizer(This) -> boolean()

Types:

This = wxSizerItem()

See **external documentation**.

isShown(This) -> boolean()

Types:

This = wxSizerItem()

See **external documentation**.

isSpacer(This) -> boolean()

Types:

This = wxSizerItem()

See **external documentation**.

isWindow(This) -> boolean()

Types:

This = wxSizerItem()

See **external documentation**.

setBorder(This, Border) -> ok

Types:

This = wxSizerItem()
Border = integer()

See **external documentation**.

setDimension(This, Pos, Size) -> ok

Types:

This = wxSizerItem()
Pos = {X::integer(), Y::integer()}
Size = {W::integer(), H::integer()}

See **external documentation**.

setFlag(This, Flag) -> ok

Types:

This = wxSizerItem()
Flag = integer()

See **external documentation**.

setInitSize(This, X, Y) -> ok

Types:

This = wxSizerItem()
X = integer()
Y = integer()

See **external documentation**.

setMinSize(This, Size) -> ok

Types:

```
This = wxSizerItem()
Size = {W::integer(), H::integer()}
```

See **external documentation**.

setMinSize(This, X, Y) -> ok

Types:

```
This = wxSizerItem()
X = integer()
Y = integer()
```

See **external documentation**.

setProportion(This, Proportion) -> ok

Types:

```
This = wxSizerItem()
Proportion = integer()
```

See **external documentation**.

setRatio(This, Ratio) -> ok

Types:

```
This = wxSizerItem()
Ratio = number()
```

See **external documentation**.
Also:
setRatio(This, Size) -> ok when
This::wxSizerItem(), Size::{W::integer(), H::integer()}.

setRatio(This, Width, Height) -> ok

Types:

```
This = wxSizerItem()
Width = integer()
Height = integer()
```

See **external documentation**.

setSizer(This, Sizer) -> ok

Types:

```
This = wxSizerItem()
Sizer = wxSizer() (see module wxSizer)
```

See **external documentation**.

setSpacer(This, Size) -> ok

Types:

```
This = wxSizerItem()
Size = {W::integer(), H::integer()}
```

See **external documentation**.

setSpacer(This, Width, Height) -> ok

Types:

```
This = wxSizerItem()
Width = integer()
Height = integer()
```

See **external documentation**.

setWindow(This, Window) -> ok

Types:

```
This = wxSizerItem()
Window = wxWindow() (see module wxWindow)
```

See **external documentation**.

show(This, Show) -> ok

Types:

```
This = wxSizerItem()
Show = boolean()
```

See **external documentation**.

destroy(This::wxSizerItem()) -> ok

Destroys this object, do not use object again

wxSlider

Erlang module

See external documentation: **wxSlider**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxSlider()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSlider()`

See **external documentation**.

`new(Parent, Id, Value, MinValue, MaxValue) -> wxSlider()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Value = integer()
MinValue = integer()
MaxValue = integer()
```

Equivalent to *new(Parent, Id, Value, MinValue, MaxValue, [])*.

`new(Parent, Id, Value, MinValue, MaxValue, Option::[Option]) -> wxSlider()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Value = integer()
MinValue = integer()
MaxValue = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

`create(This, Parent, Id, Value, MinValue, MaxValue) -> boolean()`

Types:

```
This = wxSlider()
```

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Value = integer()
MinValue = integer()
MaxValue = integer()
```

Equivalent to *create(This, Parent, Id, Value, MinValue, MaxValue, [])*.

create(This, Parent, Id, Value, MinValue, MaxValue, Option::[Option]) -> boolean()

Types:

```
This = wxSlider()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Value = integer()
MinValue = integer()
MaxValue = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

getLineSize(This) -> integer()

Types:

```
This = wxSlider()
```

See **external documentation**.

getMax(This) -> integer()

Types:

```
This = wxSlider()
```

See **external documentation**.

getMin(This) -> integer()

Types:

```
This = wxSlider()
```

See **external documentation**.

getPageSize(This) -> integer()

Types:

```
This = wxSlider()
```

See **external documentation**.

getThumbLength(This) -> integer()

Types:

```
This = wxSlider()
```

See **external documentation**.

getValue(This) -> integer()

Types:

```
This = wxSlider()
```

See **external documentation**.

setLineSize(This, LineSize) -> ok

Types:

```
This = wxSlider()
LineSize = integer()
```

See **external documentation**.

setPageSize(This, PageSize) -> ok

Types:

```
This = wxSlider()
PageSize = integer()
```

See **external documentation**.

setRange(This, MinValue, MaxValue) -> ok

Types:

```
This = wxSlider()
MinValue = integer()
MaxValue = integer()
```

See **external documentation**.

setThumbLength(This, LenPixels) -> ok

Types:

```
This = wxSlider()
LenPixels = integer()
```

See **external documentation**.

setValue(This, Value) -> ok

Types:

```
This = wxSlider()
Value = integer()
```

See **external documentation**.

destroy(This::wxSlider()) -> ok

Destroys this object, do not use object again

wxSpinButton

Erlang module

See external documentation: **wxSpinButton**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxSpinButton()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSpinButton()`

See **external documentation**.

`new(Parent) -> wxSpinButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxSpinButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

```
This = wxSpinButton()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

```
This = wxSpinButton()
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

getMax(This) -> integer()

Types:

This = wxSpinButton()

See **external documentation**.

getMin(This) -> integer()

Types:

This = wxSpinButton()

See **external documentation**.

getValue(This) -> integer()

Types:

This = wxSpinButton()

See **external documentation**.

setRange(This, MinVal, MaxVal) -> ok

Types:

This = wxSpinButton()
MinVal = integer()
MaxVal = integer()

See **external documentation**.

setValue(This, Value) -> ok

Types:

This = wxSpinButton()
Value = integer()

See **external documentation**.

destroy(This::wxSpinButton()) -> ok

Destroys this object, do not use object again

wxSpinCtrl

Erlang module

See external documentation: **wxSpinCtrl**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxSpinCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSpinCtrl()`

See **external documentation**.

`new(Parent) -> wxSpinCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxSpinCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {value, chardata() (see module unicode)} |
{pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}}
| {style, integer()} | {min, integer()} | {max, integer()} | {initial,
integer()}
```

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

```
This = wxSpinCtrl()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

```
This = wxSpinCtrl()
Parent = wxWindow() (see module wxWindow)
```

```
Option = {id, integer()} | {value, chardata() (see module unicode)} |
{pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}}
| {style, integer()} | {min, integer()} | {max, integer()} | {initial,
integer()}
```

See **external documentation**.

setValue(This, Value) -> ok

Types:

```
This = wxSpinCtrl()
Value = integer()
```

See **external documentation**.
Also:
setValue(This, Text) -> ok when
This::wxSpinCtrl(), Text::unicode:chardata().

getValue(This) -> integer()

Types:

```
This = wxSpinCtrl()
```

See **external documentation**.

setRange(This, MinVal, MaxVal) -> ok

Types:

```
This = wxSpinCtrl()
MinVal = integer()
MaxVal = integer()
```

See **external documentation**.

setSelection(This, From, To) -> ok

Types:

```
This = wxSpinCtrl()
From = integer()
To = integer()
```

See **external documentation**.

getMin(This) -> integer()

Types:

```
This = wxSpinCtrl()
```

See **external documentation**.

getMax(This) -> integer()

Types:

```
This = wxSpinCtrl()
```

See **external documentation**.

destroy(This::wxSpinCtrl()) -> ok

Destroys this object, do not use object again

wxSpinEvent

Erlang module

See external documentation: **wxSpinEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_spinctrl_updated, *spin_up*, *spin_down*, *spin*

See also the message variant *#wxSpin{}* event record type.

This class is derived (and can use functions) from:
wxNotifyEvent
wxCommandEvent
wxEvent

DATA TYPES

`wxSpinEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getPosition(This) -> integer()`

Types:

`This = wxSpinEvent()`

See **external documentation**.

`setPosition(This, Pos) -> ok`

Types:

`This = wxSpinEvent()`
`Pos = integer()`

See **external documentation**.

wxSplashScreen

Erlang module

See external documentation: **wxSplashScreen**.

This class is derived (and can use functions) from:
wxFrame
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxSplashScreen()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSplashScreen()`

See **external documentation**.

`new(Bitmap, SplashStyle, Milliseconds, Parent, Id) -> wxSplashScreen()`

Types:

```
Bitmap = wxBitmap() (see module wxBitmap)
SplashStyle = integer()
Milliseconds = integer()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Bitmap, SplashStyle, Milliseconds, Parent, Id, [])*.

`new(Bitmap, SplashStyle, Milliseconds, Parent, Id, Option::[Option]) -> wxSplashScreen()`

Types:

```
Bitmap = wxBitmap() (see module wxBitmap)
SplashStyle = integer()
Milliseconds = integer()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`getSplashStyle(This) -> integer()`

Types:

```
This = wxSplashScreen()
```

See **external documentation**.

getTimeout(This) -> integer()

Types:

```
This = wxSplashScreen()
```

See **external documentation**.

destroy(This::wxSplashScreen()) -> ok

Destroys this object, do not use object again

wxSplitterEvent

Erlang module

See external documentation: **wxSplitterEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_splitter_sash_pos_changed, *command_splitter_sash_pos_changing*, *command_splitter_doubleclicked*, *command_splitter_unsplit*

See also the message variant *#wxSplitter{}* event record type.

This class is derived (and can use functions) from:
wxNotifyEvent
wxCommandEvent
wxEvent

DATA TYPES

`wxSplitterEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getSashPosition(This) -> integer()`

Types:

`This = wxSplitterEvent()`

See **external documentation**.

`getX(This) -> integer()`

Types:

`This = wxSplitterEvent()`

See **external documentation**.

`getY(This) -> integer()`

Types:

`This = wxSplitterEvent()`

See **external documentation**.

`getWindowBeingRemoved(This) -> wxWindow() (see module wxWindow)`

Types:

`This = wxSplitterEvent()`

See **external documentation**.

`setSashPosition(This, Pos) -> ok`

Types:

```
This = wxSplitterEvent()
Pos = integer()
```

See **external documentation**.

wxSplitterWindow

Erlang module

See external documentation: **wxSplitterWindow**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxSplitterWindow()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxSplitterWindow()`

See **external documentation**.

`new(Parent) -> wxSplitterWindow()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxSplitterWindow()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

```
This = wxSplitterWindow()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

```
This = wxSplitterWindow()
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

getMinimumPaneSize(This) -> integer()

Types:

This = wxSplitterWindow()

See **external documentation**.

getSashGravity(This) -> number()

Types:

This = wxSplitterWindow()

See **external documentation**.

getSashPosition(This) -> integer()

Types:

This = wxSplitterWindow()

See **external documentation**.

getSplitMode(This) -> wx_enum() (see module wx)

Types:

This = wxSplitterWindow()

See **external documentation**.
Res = ?wxSPLIT_HORIZONTAL | ?wxSPLIT_VERTICAL

getWindow1(This) -> wxWindow() (see module wxWindow)

Types:

This = wxSplitterWindow()

See **external documentation**.

getWindow2(This) -> wxWindow() (see module wxWindow)

Types:

This = wxSplitterWindow()

See **external documentation**.

initialize(This, Window) -> ok

Types:

This = wxSplitterWindow()

Window = wxWindow() (see module wxWindow)

See **external documentation**.

isSplit(This) -> boolean()

Types:

This = wxSplitterWindow()

See **external documentation**.

replaceWindow(This, WinOld, WinNew) -> boolean()

Types:

This = wxSplitterWindow()
WinOld = wxWindow() (see module wxWindow)
WinNew = wxWindow() (see module wxWindow)

See **external documentation**.

setSashGravity(This, Gravity) -> ok

Types:

This = wxSplitterWindow()
Gravity = number()

See **external documentation**.

setSashPosition(This, Position) -> ok

Types:

This = wxSplitterWindow()
Position = integer()

Equivalent to *setSashPosition(This, Position, [])*.

setSashPosition(This, Position, Option::[Option]) -> ok

Types:

This = wxSplitterWindow()
Position = integer()
Option = {redraw, boolean()}

See **external documentation**.

setSashSize(This, Width) -> ok

Types:

This = wxSplitterWindow()
Width = integer()

See **external documentation**.

setMinimumPaneSize(This, Min) -> ok

Types:

This = wxSplitterWindow()
Min = integer()

See **external documentation**.

setSplitMode(This, Mode) -> ok

Types:

This = wxSplitterWindow()
Mode = integer()

See **external documentation**.

splitHorizontally(This, Window1, Window2) -> boolean()

Types:

```
This = wxSplitterWindow()
Window1 = wxWindow() (see module wxWindow)
Window2 = wxWindow() (see module wxWindow)
```

Equivalent to *splitHorizontally(This, Window1, Window2, [])*.

splitHorizontally(This, Window1, Window2, Option::[Option]) -> boolean()

Types:

```
This = wxSplitterWindow()
Window1 = wxWindow() (see module wxWindow)
Window2 = wxWindow() (see module wxWindow)
Option = {sashPosition, integer()}
```

See **external documentation**.

splitVertically(This, Window1, Window2) -> boolean()

Types:

```
This = wxSplitterWindow()
Window1 = wxWindow() (see module wxWindow)
Window2 = wxWindow() (see module wxWindow)
```

Equivalent to *splitVertically(This, Window1, Window2, [])*.

splitVertically(This, Window1, Window2, Option::[Option]) -> boolean()

Types:

```
This = wxSplitterWindow()
Window1 = wxWindow() (see module wxWindow)
Window2 = wxWindow() (see module wxWindow)
Option = {sashPosition, integer()}
```

See **external documentation**.

unsplit(This) -> boolean()

Types:

```
This = wxSplitterWindow()
```

Equivalent to *unsplit(This, [])*.

unsplit(This, Option::[Option]) -> boolean()

Types:

```
This = wxSplitterWindow()
Option = {toRemove, wxWindow() (see module wxWindow)}
```

See **external documentation**.

updateSize(This) -> ok

Types:

```
This = wxSplitterWindow()
```

See **external documentation**.

destroy(This::wxSplitterWindow()) -> ok

Destroys this object, do not use object again

wxStaticBitmap

Erlang module

See external documentation: **wxStaticBitmap**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxStaticBitmap()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxStaticBitmap()`

See **external documentation**.

`new(Parent, Id, Label) -> wxStaticBitmap()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = wxBitmap() (see module wxBitmap)
```

Equivalent to *new(Parent, Id, Label, [])*.

`new(Parent, Id, Label, Option::[Option]) -> wxStaticBitmap()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = wxBitmap() (see module wxBitmap)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent, Id, Label) -> boolean()`

Types:

```
This = wxStaticBitmap()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = wxBitmap() (see module wxBitmap)
```

Equivalent to *create(This, Parent, Id, Label, [])*.

```
create(This, Parent, Id, Label, Option::[Option]) -> boolean()
```

Types:

```
This = wxStaticBitmap()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = wxBitmap() (see module wxBitmap)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

```
getBitmap(This) -> wxBitmap() (see module wxBitmap)
```

Types:

```
This = wxStaticBitmap()
```

See **external documentation**.

```
setBitmap(This, Bitmap) -> ok
```

Types:

```
This = wxStaticBitmap()
Bitmap = wxBitmap() (see module wxBitmap)
```

See **external documentation**.

```
destroy(This::wxStaticBitmap()) -> ok
```

Destroys this object, do not use object again

wxStaticBox

Erlang module

See external documentation: **wxStaticBox**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxStaticBox()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxStaticBox()`

See **external documentation**.

`new(Parent, Id, Label) -> wxStaticBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Label, [])*.

`new(Parent, Id, Label, Option::[Option]) -> wxStaticBox()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent, Id, Label) -> boolean()`

Types:

```
This = wxStaticBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Label, [])*.

```
create(This, Parent, Id, Label, Option::[Option]) -> boolean()
```

Types:

```
This = wxStaticBox()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

```
destroy(This::wxStaticBox()) -> ok
```

Destroys this object, do not use object again

wxStaticBoxSizer

Erlang module

See external documentation: **wxStaticBoxSizer**.

This class is derived (and can use functions) from:
wxBoxSizer
wxSizer

DATA TYPES

`wxStaticBoxSizer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Orient, Win) -> wxStaticBoxSizer()`

Types:

```
Orient = integer()
Win = wxWindow() (see module wxWindow)
```

See **external documentation**.
Also:
new(Box, Orient) -> wxStaticBoxSizer() when
Box::wxStaticBox:wxStaticBox(), Orient::integer().

`new(Orient, Win, Option::[Option]) -> wxStaticBoxSizer()`

Types:

```
Orient = integer()
Win = wxWindow() (see module wxWindow)
Option = {label, chardata() (see module unicode)}
```

See **external documentation**.

`getStaticBox(This) -> wxStaticBox() (see module wxStaticBox)`

Types:

```
This = wxStaticBoxSizer()
```

See **external documentation**.

`destroy(This::wxStaticBoxSizer()) -> ok`

Destroys this object, do not use object again

wxStaticLine

Erlang module

See external documentation: **wxStaticLine**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxStaticLine()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxStaticLine()`

See **external documentation**.

`new(Parent) -> wxStaticLine()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxStaticLine()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

```
This = wxStaticLine()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

```
This = wxStaticLine()
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size,
{W::integer(), H::integer()}} | {style, integer()}
```

See **external documentation**.

isVertical(This) -> boolean()

Types:

This = wxStaticLine()

See **external documentation**.

getDefaultSize() -> integer()

See **external documentation**.

destroy(This::wxStaticLine()) -> ok

Destroys this object, do not use object again

wxStaticText

Erlang module

See external documentation: **wxStaticText**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxStaticText()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxStaticText()`

See **external documentation**.

`new(Parent, Id, Label) -> wxStaticText()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Label, [])*.

`new(Parent, Id, Label, Option::[Option]) -> wxStaticText()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`create(This, Parent, Id, Label) -> boolean()`

Types:

```
This = wxStaticText()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Label, [])*.

create(This, Parent, Id, Label, Option::[Option]) -> boolean()

Types:

This = wxStaticText()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Label = chardata() (see module unicode)

Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}

See **external documentation**.

getLabel(This) -> charlist() (see module unicode)

Types:

This = wxStaticText()

See **external documentation**.

setLabel(This, Label) -> ok

Types:

This = wxStaticText()

Label = chardata() (see module unicode)

See **external documentation**.

wrap(This, Width) -> ok

Types:

This = wxStaticText()

Width = integer()

See **external documentation**.

destroy(This::wxStaticText()) -> ok

Destroys this object, do not use object again

wxStatusBar

Erlang module

See external documentation: **wxStatusBar**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxStatusBar()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxStatusBar()`

See **external documentation**.

`new(Parent) -> wxStatusBar()`

Types:

```
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxStatusBar()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Option = {winid, integer()} | {style, integer()}
```

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

```
This = wxStatusBar()
Parent = wxWindow() (see module wxWindow)
```

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

```
This = wxStatusBar()
Parent = wxWindow() (see module wxWindow)
Option = {winid, integer()} | {style, integer()}
```

See **external documentation**.

getFieldRect(This, I, Rect) -> boolean()

Types:

This = wxStatusBar()
I = integer()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}

See **external documentation**.

getFieldsCount(This) -> integer()

Types:

This = wxStatusBar()

See **external documentation**.

getStatusText(This) -> charlist() (see module unicode)

Types:

This = wxStatusBar()

Equivalent to *getStatusText(This, [])*.

getStatusText(This, Option::[Option]) -> charlist() (see module unicode)

Types:

This = wxStatusBar()
Option = {number, integer()}

See **external documentation**.

popStatusText(This) -> ok

Types:

This = wxStatusBar()

Equivalent to *popStatusText(This, [])*.

popStatusText(This, Option::[Option]) -> ok

Types:

This = wxStatusBar()
Option = {number, integer()}

See **external documentation**.

pushStatusText(This, Text) -> ok

Types:

This = wxStatusBar()
Text = chardata() (see module unicode)

Equivalent to *pushStatusText(This, Text, [])*.

pushStatusText(This, Text, Option::[Option]) -> ok

Types:

This = wxStatusBar()

```
Text = chardata() (see module unicode)
Option = {number, integer()}
```

See **external documentation**.

setFieldsCount(This, Number) -> ok

Types:

```
This = wxStatusBar()
Number = integer()
```

Equivalent to *setFieldsCount(This, Number, [])*.

setFieldsCount(This, Number, Option::[Option]) -> ok

Types:

```
This = wxStatusBar()
Number = integer()
Option = {widths, [integer()]}
```

See **external documentation**.

setMinHeight(This, Height) -> ok

Types:

```
This = wxStatusBar()
Height = integer()
```

See **external documentation**.

setStatusText(This, Text) -> ok

Types:

```
This = wxStatusBar()
Text = chardata() (see module unicode)
```

Equivalent to *setStatusText(This, Text, [])*.

setStatusText(This, Text, Option::[Option]) -> ok

Types:

```
This = wxStatusBar()
Text = chardata() (see module unicode)
Option = {number, integer()}
```

See **external documentation**.

setStatusWidths(This, Widths_field) -> ok

Types:

```
This = wxStatusBar()
Widths_field = [integer()]
```

See **external documentation**.

setStatusStyles(This, Styles) -> ok

Types:

```
This = wxStatusBar()
Styles = [integer()]
```

See **external documentation**.

destroy(This::wxStatusBar()) -> ok

Destroys this object, do not use object again

wxStdDialogButtonSizer

Erlang module

See external documentation: **wxStdDialogButtonSizer**.

This class is derived (and can use functions) from:
wxBoxSizer
wxSizer

DATA TYPES

`wxStdDialogButtonSizer()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxStdDialogButtonSizer()`

See **external documentation**.

`addButton(This, Button) -> ok`

Types:

```
This = wxStdDialogButtonSizer()
Button = wxButton() (see module wxButton)
```

See **external documentation**.

`realize(This) -> ok`

Types:

```
This = wxStdDialogButtonSizer()
```

See **external documentation**.

`setAffirmativeButton(This, Button) -> ok`

Types:

```
This = wxStdDialogButtonSizer()
Button = wxButton() (see module wxButton)
```

See **external documentation**.

`setCancelButton(This, Button) -> ok`

Types:

```
This = wxStdDialogButtonSizer()
Button = wxButton() (see module wxButton)
```

See **external documentation**.

setNegativeButton(This, Button) -> ok

Types:

```
This = wxStdDialogButtonSizer()
Button = wxButton() (see module wxButton)
```

See **external documentation**.

destroy(This::wxStdDialogButtonSizer()) -> ok

Destroys this object, do not use object again

wxStyledTextCtrl

Erlang module

See external documentation: **wxStyledTextCtrl**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxStyledTextCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxStyledTextCtrl()`

See **external documentation**.

`new(Parent) -> wxStyledTextCtrl()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxStyledTextCtrl()`

Types:

`Parent = wxWindow() (see module wxWindow)`

`Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}`

See **external documentation**.

`create(This, Parent) -> boolean()`

Types:

`This = wxStyledTextCtrl()`

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *create(This, Parent, [])*.

`create(This, Parent, Option::[Option]) -> boolean()`

Types:

`This = wxStyledTextCtrl()`

`Parent = wxWindow() (see module wxWindow)`

`Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}`

See **external documentation**.

addText(This, Text) -> ok

Types:

```
This = wxStyledTextCtrl()
Text = chardata() (see module unicode)
```

See **external documentation**.

addStyledText(This, Data) -> ok

Types:

```
This = wxStyledTextCtrl()
Data = wx_object() (see module wx)
```

See **external documentation**.

insertText(This, Pos, Text) -> ok

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
Text = chardata() (see module unicode)
```

See **external documentation**.

clearAll(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

clearDocumentStyle(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getLength(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getCharAt(This, Pos) -> integer()

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
```

See **external documentation**.

getCurrentPos(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getAnchor(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getStyleAt(This, Pos) -> integer()

Types:

This = wxStyledTextCtrl()

Pos = integer()

See **external documentation**.

redo(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setUndoCollection(This, CollectUndo) -> ok

Types:

This = wxStyledTextCtrl()

CollectUndo = boolean()

See **external documentation**.

selectAll(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setSavePoint(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getStyledText(This, StartPos, EndPos) -> wx_object() (see module wx)

Types:

This = wxStyledTextCtrl()

StartPos = integer()

EndPos = integer()

See **external documentation**.

canRedo(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

markerLineFromHandle(This, Handle) -> integer()

Types:

This = wxStyledTextCtrl()
Handle = integer()

See **external documentation**.

markerDeleteHandle(This, Handle) -> ok

Types:

This = wxStyledTextCtrl()
Handle = integer()

See **external documentation**.

getUndoCollection(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getViewWhiteSpace(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setViewWhiteSpace(This, ViewWS) -> ok

Types:

This = wxStyledTextCtrl()
ViewWS = integer()

See **external documentation**.

positionFromPoint(This, Pt) -> integer()

Types:

This = wxStyledTextCtrl()
Pt = {X::integer(), Y::integer()}

See **external documentation**.

positionFromPointClose(This, X, Y) -> integer()

Types:

This = wxStyledTextCtrl()
X = integer()

```
   Y = integer()
```

See **external documentation**.

gotoLine(This, Line) -> ok

Types:

```
   This = wxStyledTextCtrl()
   Line = integer()
```

See **external documentation**.

gotoPos(This, Pos) -> ok

Types:

```
   This = wxStyledTextCtrl()
   Pos = integer()
```

See **external documentation**.

setAnchor(This, PosAnchor) -> ok

Types:

```
   This = wxStyledTextCtrl()
   PosAnchor = integer()
```

See **external documentation**.

getCurLine(This) -> Result

Types:

```
   Result = {Res::charlist() (see module unicode), LinePos::integer()}
   This = wxStyledTextCtrl()
```

See **external documentation**.

getEndStyled(This) -> integer()

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

convertEOLs(This, EolMode) -> ok

Types:

```
   This = wxStyledTextCtrl()
   EolMode = integer()
```

See **external documentation**.

getEOLMode(This) -> integer()

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

setEOLMode(This, EolMode) -> ok

Types:

```
This = wxStyledTextCtrl()
EolMode = integer()
```

See **external documentation**.

startStyling(This, Pos, Mask) -> ok

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
Mask = integer()
```

See **external documentation**.

setStyling(This, Length, Style) -> ok

Types:

```
This = wxStyledTextCtrl()
Length = integer()
Style = integer()
```

See **external documentation**.

getBufferedDraw(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setBufferedDraw(This, Buffered) -> ok

Types:

```
This = wxStyledTextCtrl()
Buffered = boolean()
```

See **external documentation**.

setTabWidth(This, TabWidth) -> ok

Types:

```
This = wxStyledTextCtrl()
TabWidth = integer()
```

See **external documentation**.

getTabWidth(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setCodePage(This, CodePage) -> ok

Types:

```
This = wxStyledTextCtrl()
CodePage = integer()
```

See **external documentation**.

markerDefine(This, MarkerNumber, MarkerSymbol) -> ok

Types:

```
This = wxStyledTextCtrl()
MarkerNumber = integer()
MarkerSymbol = integer()
```

Equivalent to *markerDefine(This, MarkerNumber, MarkerSymbol, [])*.

markerDefine(This, MarkerNumber, MarkerSymbol, Option::[Option]) -> ok

Types:

```
This = wxStyledTextCtrl()
MarkerNumber = integer()
MarkerSymbol = integer()
Option = {foreground, wx_colour() (see module wx)} | {background,
wx_colour() (see module wx)}
```

See **external documentation**.

markerSetForeground(This, MarkerNumber, Fore) -> ok

Types:

```
This = wxStyledTextCtrl()
MarkerNumber = integer()
Fore = wx_colour() (see module wx)
```

See **external documentation**.

markerSetBackground(This, MarkerNumber, Back) -> ok

Types:

```
This = wxStyledTextCtrl()
MarkerNumber = integer()
Back = wx_colour() (see module wx)
```

See **external documentation**.

markerAdd(This, Line, MarkerNumber) -> integer()

Types:

```
This = wxStyledTextCtrl()
Line = integer()
MarkerNumber = integer()
```

See **external documentation**.

```
markerDelete(This, Line, MarkerNumber) -> ok
```

Types:

```
This = wxStyledTextCtrl()
Line = integer()
MarkerNumber = integer()
```

See **external documentation**.

```
markerDeleteAll(This, MarkerNumber) -> ok
```

Types:

```
This = wxStyledTextCtrl()
MarkerNumber = integer()
```

See **external documentation**.

```
markerGet(This, Line) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
Line = integer()
```

See **external documentation**.

```
markerNext(This, LineStart, MarkerMask) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
LineStart = integer()
MarkerMask = integer()
```

See **external documentation**.

```
markerPrevious(This, LineStart, MarkerMask) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
LineStart = integer()
MarkerMask = integer()
```

See **external documentation**.

```
markerDefineBitmap(This, MarkerNumber, Bmp) -> ok
```

Types:

```
This = wxStyledTextCtrl()
MarkerNumber = integer()
Bmp = wxBitmap() (see module wxBitmap)
```

See **external documentation**.

```
markerAddSet(This, Line, Set) -> ok
```

Types:

```
This = wxStyledTextCtrl()
```

```
Line = integer()
Set = integer()
```

See **external documentation**.

markerSetAlpha(This, MarkerNumber, Alpha) -> ok

Types:

```
This = wxStyledTextCtrl()
MarkerNumber = integer()
Alpha = integer()
```

See **external documentation**.

setMarginType(This, Margin, MarginType) -> ok

Types:

```
This = wxStyledTextCtrl()
Margin = integer()
MarginType = integer()
```

See **external documentation**.

getMarginType(This, Margin) -> integer()

Types:

```
This = wxStyledTextCtrl()
Margin = integer()
```

See **external documentation**.

setMarginWidth(This, Margin, PixelWidth) -> ok

Types:

```
This = wxStyledTextCtrl()
Margin = integer()
PixelWidth = integer()
```

See **external documentation**.

getMarginWidth(This, Margin) -> integer()

Types:

```
This = wxStyledTextCtrl()
Margin = integer()
```

See **external documentation**.

setMarginMask(This, Margin, Mask) -> ok

Types:

```
This = wxStyledTextCtrl()
Margin = integer()
Mask = integer()
```

See **external documentation**.

getMarginMask(This, Margin) -> integer()

Types:

This = wxStyledTextCtrl()
Margin = integer()

See **external documentation**.

setMarginSensitive(This, Margin, Sensitive) -> ok

Types:

This = wxStyledTextCtrl()
Margin = integer()
Sensitive = boolean()

See **external documentation**.

getMarginSensitive(This, Margin) -> boolean()

Types:

This = wxStyledTextCtrl()
Margin = integer()

See **external documentation**.

styleClearAll(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

styleSetForeground(This, Style, Fore) -> ok

Types:

This = wxStyledTextCtrl()
Style = integer()
Fore = wx_colour() (see module wx)

See **external documentation**.

styleSetBackground(This, Style, Back) -> ok

Types:

This = wxStyledTextCtrl()
Style = integer()
Back = wx_colour() (see module wx)

See **external documentation**.

styleSetBold(This, Style, Bold) -> ok

Types:

This = wxStyledTextCtrl()
Style = integer()
Bold = boolean()

See **external documentation**.

styleSetItalic(This, Style, Italic) -> ok

Types:

```
This = wxStyledTextCtrl()
Style = integer()
Italic = boolean()
```

See **external documentation**.

styleSetSize(This, Style, SizePoints) -> ok

Types:

```
This = wxStyledTextCtrl()
Style = integer()
SizePoints = integer()
```

See **external documentation**.

styleSetFaceName(This, Style, FontName) -> ok

Types:

```
This = wxStyledTextCtrl()
Style = integer()
FontName = chardata() (see module unicode)
```

See **external documentation**.

styleSetEOLFilled(This, Style, Filled) -> ok

Types:

```
This = wxStyledTextCtrl()
Style = integer()
Filled = boolean()
```

See **external documentation**.

styleResetDefault(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

styleSetUnderline(This, Style, Underline) -> ok

Types:

```
This = wxStyledTextCtrl()
Style = integer()
Underline = boolean()
```

See **external documentation**.

```
styleSetCase(This, Style, CaseForce) -> ok
```

Types:

```
This = wxStyledTextCtrl()
Style = integer()
CaseForce = integer()
```

See **external documentation**.

```
styleSetHotSpot(This, Style, Hotspot) -> ok
```

Types:

```
This = wxStyledTextCtrl()
Style = integer()
Hotspot = boolean()
```

See **external documentation**.

```
setSelForeground(This, UseSetting, Fore) -> ok
```

Types:

```
This = wxStyledTextCtrl()
UseSetting = boolean()
Fore = wx_colour() (see module wx)
```

See **external documentation**.

```
setSelBackground(This, UseSetting, Back) -> ok
```

Types:

```
This = wxStyledTextCtrl()
UseSetting = boolean()
Back = wx_colour() (see module wx)
```

See **external documentation**.

```
getSelAlpha(This) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

```
setSelAlpha(This, Alpha) -> ok
```

Types:

```
This = wxStyledTextCtrl()
Alpha = integer()
```

See **external documentation**.

```
setCaretForeground(This, Fore) -> ok
```

Types:

```
This = wxStyledTextCtrl()
Fore = wx_colour() (see module wx)
```

See **external documentation**.

cmdKeyAssign(This, Key, Modifiers, Cmd) -> ok

Types:

```
This = wxStyledTextCtrl()
Key = integer()
Modifiers = integer()
Cmd = integer()
```

See **external documentation**.

cmdKeyClear(This, Key, Modifiers) -> ok

Types:

```
This = wxStyledTextCtrl()
Key = integer()
Modifiers = integer()
```

See **external documentation**.

cmdKeyClearAll(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setStyleBytes(This, Length) -> integer()

Types:

```
This = wxStyledTextCtrl()
Length = integer()
```

See **external documentation**.

styleSetVisible(This, Style, Visible) -> ok

Types:

```
This = wxStyledTextCtrl()
Style = integer()
Visible = boolean()
```

See **external documentation**.

getCaretPeriod(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setCaretPeriod(This, PeriodMilliseconds) -> ok

Types:

```
This = wxStyledTextCtrl()
```

```
PeriodMilliseconds = integer()
```

See **external documentation**.

setWordChars(This, Characters) -> ok

Types:

```
This = wxStyledTextCtrl()
Characters = chardata() (see module unicode)
```

See **external documentation**.

beginUndoAction(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

endUndoAction(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

indicatorSetStyle(This, Indic, Style) -> ok

Types:

```
This = wxStyledTextCtrl()
Indic = integer()
Style = integer()
```

See **external documentation**.

indicatorGetStyle(This, Indic) -> integer()

Types:

```
This = wxStyledTextCtrl()
Indic = integer()
```

See **external documentation**.

indicatorSetForeground(This, Indic, Fore) -> ok

Types:

```
This = wxStyledTextCtrl()
Indic = integer()
Fore = wx_colour() (see module wx)
```

See **external documentation**.

indicatorGetForeground(This, Indic) -> wx_colour4() (see module wx)

Types:

```
This = wxStyledTextCtrl()
Indic = integer()
```

See **external documentation**.

setWhitespaceForeground(This, UseSetting, Fore) -> ok

Types:

```
This = wxStyledTextCtrl()
UseSetting = boolean()
Fore = wx_colour() (see module wx)
```

See **external documentation**.

setWhitespaceBackground(This, UseSetting, Back) -> ok

Types:

```
This = wxStyledTextCtrl()
UseSetting = boolean()
Back = wx_colour() (see module wx)
```

See **external documentation**.

getStyleBits(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setLineState(This, Line, State) -> ok

Types:

```
This = wxStyledTextCtrl()
Line = integer()
State = integer()
```

See **external documentation**.

getLineState(This, Line) -> integer()

Types:

```
This = wxStyledTextCtrl()
Line = integer()
```

See **external documentation**.

getMaxLineState(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getCaretLineVisible(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setCaretLineVisible(This, Show) -> ok

Types:

This = wxStyledTextCtrl()

Show = boolean()

See **external documentation**.

getCaretLineBackground(This) -> wx_colour4() (see module wx)

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setCaretLineBackground(This, Back) -> ok

Types:

This = wxStyledTextCtrl()

Back = wx_colour() (see module wx)

See **external documentation**.

autoCompShow(This, LenEntered, ItemList) -> ok

Types:

This = wxStyledTextCtrl()

LenEntered = integer()

ItemList = chardata() (see module unicode)

See **external documentation**.

autoCompCancel(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompActive(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompPosStart(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompComplete(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompStops(This, CharacterSet) -> ok

Types:

This = wxStyledTextCtrl()
CharacterSet = chardata() (see module unicode)

See **external documentation**.

autoCompSetSeparator(This, SeparatorCharacter) -> ok

Types:

This = wxStyledTextCtrl()
SeparatorCharacter = integer()

See **external documentation**.

autoCompGetSeparator(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompSelect(This, Text) -> ok

Types:

This = wxStyledTextCtrl()
Text = chardata() (see module unicode)

See **external documentation**.

autoCompSetCancelAtStart(This, Cancel) -> ok

Types:

This = wxStyledTextCtrl()
Cancel = boolean()

See **external documentation**.

autoCompGetCancelAtStart(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompSetFillUps(This, CharacterSet) -> ok

Types:

This = wxStyledTextCtrl()
CharacterSet = chardata() (see module unicode)

See **external documentation**.

autoCompSetChooseSingle(This, ChooseSingle) -> ok

Types:

This = wxStyledTextCtrl()

```
   ChooseSingle = boolean()
```

See **external documentation**.

autoCompGetChooseSingle(This) -> boolean()

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

autoCompSetIgnoreCase(This, IgnoreCase) -> ok

Types:

```
   This = wxStyledTextCtrl()
   IgnoreCase = boolean()
```

See **external documentation**.

autoCompGetIgnoreCase(This) -> boolean()

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

userListShow(This, ListType, ItemList) -> ok

Types:

```
   This = wxStyledTextCtrl()
   ListType = integer()
   ItemList = chardata() (see module unicode)
```

See **external documentation**.

autoCompSetAutoHide(This, AutoHide) -> ok

Types:

```
   This = wxStyledTextCtrl()
   AutoHide = boolean()
```

See **external documentation**.

autoCompGetAutoHide(This) -> boolean()

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

autoCompSetDropRestOfWord(This, DropRestOfWord) -> ok

Types:

```
   This = wxStyledTextCtrl()
   DropRestOfWord = boolean()
```

See **external documentation**.

autoCompGetDropRestOfWord(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

registerImage(This, Type, Bmp) -> ok

Types:

This = wxStyledTextCtrl()
Type = integer()
Bmp = wxBitmap() (see module wxBitmap)

See **external documentation**.

clearRegisteredImages(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompGetTypeSeparator(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompSetTypeSeparator(This, SeparatorCharacter) -> ok

Types:

This = wxStyledTextCtrl()
SeparatorCharacter = integer()

See **external documentation**.

autoCompSetMaxWidth(This, CharacterCount) -> ok

Types:

This = wxStyledTextCtrl()
CharacterCount = integer()

See **external documentation**.

autoCompGetMaxWidth(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompSetMaxHeight(This, RowCount) -> ok

Types:

This = wxStyledTextCtrl()
RowCount = integer()

See **external documentation**.

autoCompGetMaxHeight(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setIndent(This, IndentSize) -> ok

Types:

This = wxStyledTextCtrl()

IndentSize = integer()

See **external documentation**.

getIndent(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setUseTabs(This, UseTabs) -> ok

Types:

This = wxStyledTextCtrl()

UseTabs = boolean()

See **external documentation**.

getUseTabs(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setLineIndentation(This, Line, IndentSize) -> ok

Types:

This = wxStyledTextCtrl()

Line = integer()

IndentSize = integer()

See **external documentation**.

getLineIndentation(This, Line) -> integer()

Types:

This = wxStyledTextCtrl()

Line = integer()

See **external documentation**.

getLineIndentPosition(This, Line) -> integer()

Types:

This = wxStyledTextCtrl()

Line = integer()

See **external documentation**.

getColumn(This, Pos) -> integer()

Types:

This = wxStyledTextCtrl()

Pos = integer()

See **external documentation**.

setUseHorizontalScrollBar(This, Show) -> ok

Types:

This = wxStyledTextCtrl()

Show = boolean()

See **external documentation**.

getUseHorizontalScrollBar(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setIndentationGuides(This, Show) -> ok

Types:

This = wxStyledTextCtrl()

Show = boolean()

See **external documentation**.

getIndentationGuides(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setHighlightGuide(This, Column) -> ok

Types:

This = wxStyledTextCtrl()

Column = integer()

See **external documentation**.

getHighlightGuide(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getLineEndPosition(This, Line) -> integer()

Types:

```
This = wxStyledTextCtrl()
Line = integer()
```

See **external documentation**.

getCodePage(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getCaretForeground(This) -> wx_colour4() (see module wx)

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getReadOnly(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setCurrentPos(This, Pos) -> ok

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
```

See **external documentation**.

setSelectionStart(This, Pos) -> ok

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
```

See **external documentation**.

getSelectionStart(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setSelectionEnd(This, Pos) -> ok

Types:

```
This = wxStyledTextCtrl()
```

```
Pos = integer()
```

See **external documentation**.

```
getSelectionEnd(This) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

```
setPrintMagnification(This, Magnification) -> ok
```

Types:

```
This = wxStyledTextCtrl()
Magnification = integer()
```

See **external documentation**.

```
getPrintMagnification(This) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

```
setPrintColourMode(This, Mode) -> ok
```

Types:

```
This = wxStyledTextCtrl()
Mode = integer()
```

See **external documentation**.

```
getPrintColourMode(This) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

```
findText(This, MinPos, MaxPos, Text) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
MinPos = integer()
MaxPos = integer()
Text = chardata() (see module unicode)
```

Equivalent to *findText(This, MinPos, MaxPos, Text, [])*.

```
findText(This, MinPos, MaxPos, Text, Option::[Option]) -> integer()
```

Types:

```
This = wxStyledTextCtrl()
MinPos = integer()
MaxPos = integer()
```

```
   Text = chardata() (see module unicode)
   Option = {flags, integer()}
```

See **external documentation**.

```
formatRange(This, DoDraw, StartPos, EndPos, Draw, Target, RenderRect,
PageRect) -> integer()
```

Types:

```
   This = wxStyledTextCtrl()
   DoDraw = boolean()
   StartPos = integer()
   EndPos = integer()
   Draw = wxDC() (see module wxDC)
   Target = wxDC() (see module wxDC)
   RenderRect = {X::integer(), Y::integer(), W::integer(), H::integer()}
   PageRect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

See **external documentation**.

```
getFirstVisibleLine(This) -> integer()
```

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

```
getLine(This, Line) -> charlist() (see module unicode)
```

Types:

```
   This = wxStyledTextCtrl()
   Line = integer()
```

See **external documentation**.

```
getLineCount(This) -> integer()
```

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

```
setMarginLeft(This, PixelWidth) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
   PixelWidth = integer()
```

See **external documentation**.

```
getMarginLeft(This) -> integer()
```

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

setMarginRight(This, PixelWidth) -> ok

Types:

This = wxStyledTextCtrl()
PixelWidth = integer()

See **external documentation**.

getMarginRight(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getModify(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setSelection(This, Start, End) -> ok

Types:

This = wxStyledTextCtrl()
Start = integer()
End = integer()

See **external documentation**.

getSelectedText(This) -> charlist() (see module unicode)

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getTextRange(This, StartPos, EndPos) -> charlist() (see module unicode)

Types:

This = wxStyledTextCtrl()
StartPos = integer()
EndPos = integer()

See **external documentation**.

hideSelection(This, Normal) -> ok

Types:

This = wxStyledTextCtrl()
Normal = boolean()

See **external documentation**.

lineFromPosition(This, Pos) -> integer()

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
```

See **external documentation**.

positionFromLine(This, Line) -> integer()

Types:

```
This = wxStyledTextCtrl()
Line = integer()
```

See **external documentation**.

lineScroll(This, Columns, Lines) -> ok

Types:

```
This = wxStyledTextCtrl()
Columns = integer()
Lines = integer()
```

See **external documentation**.

ensureCaretVisible(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

replaceSelection(This, Text) -> ok

Types:

```
This = wxStyledTextCtrl()
Text = chardata() (see module unicode)
```

See **external documentation**.

setReadOnly(This, ReadOnly) -> ok

Types:

```
This = wxStyledTextCtrl()
ReadOnly = boolean()
```

See **external documentation**.

canPaste(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

canUndo(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

emptyUndoBuffer(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

undo(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

cut(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

copy(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

paste(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

clear(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setText(This, Text) -> ok

Types:

This = wxStyledTextCtrl()

Text = chardata() (see module unicode)

See **external documentation**.

getText(This) -> charlist() (see module unicode)

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getTextLength(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getOvertype(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setCaretWidth(This, PixelWidth) -> ok

Types:

```
This = wxStyledTextCtrl()
PixelWidth = integer()
```

See **external documentation**.

getCaretWidth(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setTargetStart(This, Pos) -> ok

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
```

See **external documentation**.

getTargetStart(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setTargetEnd(This, Pos) -> ok

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
```

See **external documentation**.

getTargetEnd(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

replaceTarget(This, Text) -> integer()

Types:

```
This = wxStyledTextCtrl()
Text = chardata() (see module unicode)
```

See **external documentation**.

searchInTarget(This, Text) -> integer()

Types:

```
This = wxStyledTextCtrl()
Text = chardata() (see module unicode)
```

See **external documentation**.

setSearchFlags(This, Flags) -> ok

Types:

```
This = wxStyledTextCtrl()
Flags = integer()
```

See **external documentation**.

getSearchFlags(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

callTipShow(This, Pos, Definition) -> ok

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
Definition = chardata() (see module unicode)
```

See **external documentation**.

callTipCancel(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

callTipActive(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

callTipPosAtStart(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

callTipSetHighlight(This, Start, End) -> ok

Types:

```
This = wxStyledTextCtrl()
Start = integer()
End = integer()
```

See **external documentation**.

callTipSetBackground(This, Back) -> ok

Types:

```
This = wxStyledTextCtrl()
Back = wx_colour() (see module wx)
```

See **external documentation**.

callTipSetForeground(This, Fore) -> ok

Types:

```
This = wxStyledTextCtrl()
Fore = wx_colour() (see module wx)
```

See **external documentation**.

callTipSetForegroundHighlight(This, Fore) -> ok

Types:

```
This = wxStyledTextCtrl()
Fore = wx_colour() (see module wx)
```

See **external documentation**.

callTipUseStyle(This, TabSize) -> ok

Types:

```
This = wxStyledTextCtrl()
TabSize = integer()
```

See **external documentation**.

visibleFromDocLine(This, Line) -> integer()

Types:

```
This = wxStyledTextCtrl()
Line = integer()
```

See **external documentation**.

docLineFromVisible(This, LineDisplay) -> integer()

Types:

```
This = wxStyledTextCtrl()
LineDisplay = integer()
```

See **external documentation**.

wrapCount(This, Line) -> integer()

Types:

This = wxStyledTextCtrl()
Line = integer()

See **external documentation**.

setFoldLevel(This, Line, Level) -> ok

Types:

This = wxStyledTextCtrl()
Line = integer()
Level = integer()

See **external documentation**.

getFoldLevel(This, Line) -> integer()

Types:

This = wxStyledTextCtrl()
Line = integer()

See **external documentation**.

getLastChild(This, Line, Level) -> integer()

Types:

This = wxStyledTextCtrl()
Line = integer()
Level = integer()

See **external documentation**.

getFoldParent(This, Line) -> integer()

Types:

This = wxStyledTextCtrl()
Line = integer()

See **external documentation**.

showLines(This, LineStart, LineEnd) -> ok

Types:

This = wxStyledTextCtrl()
LineStart = integer()
LineEnd = integer()

See **external documentation**.

hideLines(This, LineStart, LineEnd) -> ok

Types:

This = wxStyledTextCtrl()
LineStart = integer()

```
   LineEnd = integer()
```

See **external documentation**.

```
getLineVisible(This, Line) -> boolean()
```

Types:

```
   This = wxStyledTextCtrl()
   Line = integer()
```

See **external documentation**.

```
setFoldExpanded(This, Line, Expanded) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
   Line = integer()
   Expanded = boolean()
```

See **external documentation**.

```
getFoldExpanded(This, Line) -> boolean()
```

Types:

```
   This = wxStyledTextCtrl()
   Line = integer()
```

See **external documentation**.

```
toggleFold(This, Line) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
   Line = integer()
```

See **external documentation**.

```
ensureVisible(This, Line) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
   Line = integer()
```

See **external documentation**.

```
setFoldFlags(This, Flags) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
   Flags = integer()
```

See **external documentation**.

```
ensureVisibleEnforcePolicy(This, Line) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
```

```
   Line = integer()
```

See **external documentation**.

```
setTabIndents(This, TabIndents) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
   TabIndents = boolean()
```

See **external documentation**.

```
getTabIndents(This) -> boolean()
```

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

```
setBackSpaceUnIndents(This, BsUnIndents) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
   BsUnIndents = boolean()
```

See **external documentation**.

```
getBackSpaceUnIndents(This) -> boolean()
```

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

```
setMouseDwellTime(This, PeriodMilliseconds) -> ok
```

Types:

```
   This = wxStyledTextCtrl()
   PeriodMilliseconds = integer()
```

See **external documentation**.

```
getMouseDwellTime(This) -> integer()
```

Types:

```
   This = wxStyledTextCtrl()
```

See **external documentation**.

```
wordStartPosition(This, Pos, OnlyWordCharacters) -> integer()
```

Types:

```
   This = wxStyledTextCtrl()
   Pos = integer()
   OnlyWordCharacters = boolean()
```

See **external documentation**.

wordEndPosition(This, Pos, OnlyWordCharacters) -> integer()

Types:

This = wxStyledTextCtrl()
Pos = integer()
OnlyWordCharacters = boolean()

See **external documentation**.

setWrapMode(This, Mode) -> ok

Types:

This = wxStyledTextCtrl()
Mode = integer()

See **external documentation**.

getWrapMode(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setWrapVisualFlags(This, WrapVisualFlags) -> ok

Types:

This = wxStyledTextCtrl()
WrapVisualFlags = integer()

See **external documentation**.

getWrapVisualFlags(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setWrapVisualFlagsLocation(This, WrapVisualFlagsLocation) -> ok

Types:

This = wxStyledTextCtrl()
WrapVisualFlagsLocation = integer()

See **external documentation**.

getWrapVisualFlagsLocation(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setWrapStartIndent(This, Indent) -> ok

Types:

This = wxStyledTextCtrl()

```
Indent = integer()
```

See **external documentation**.

getWrapStartIndent(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setLayoutCache(This, Mode) -> ok

Types:

```
This = wxStyledTextCtrl()
Mode = integer()
```

See **external documentation**.

getLayoutCache(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setScrollWidth(This, PixelWidth) -> ok

Types:

```
This = wxStyledTextCtrl()
PixelWidth = integer()
```

See **external documentation**.

getScrollWidth(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

textWidth(This, Style, Text) -> integer()

Types:

```
This = wxStyledTextCtrl()
Style = integer()
Text = chardata() (see module unicode)
```

See **external documentation**.

getEndAtLastLine(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

textHeight(This, Line) -> integer()

Types:

This = wxStyledTextCtrl()

Line = integer()

See **external documentation**.

setUseVerticalScrollBar(This, Show) -> ok

Types:

This = wxStyledTextCtrl()

Show = boolean()

See **external documentation**.

getUseVerticalScrollBar(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

appendText(This, Text) -> ok

Types:

This = wxStyledTextCtrl()

Text = chardata() (see module unicode)

See **external documentation**.

getTwoPhaseDraw(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setTwoPhaseDraw(This, TwoPhase) -> ok

Types:

This = wxStyledTextCtrl()

TwoPhase = boolean()

See **external documentation**.

targetFromSelection(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

linesJoin(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

linesSplit(This, PixelWidth) -> ok

Types:

This = wxStyledTextCtrl()
PixelWidth = integer()

See **external documentation**.

setFoldMarginColour(This, UseSetting, Back) -> ok

Types:

This = wxStyledTextCtrl()
UseSetting = boolean()
Back = wx_colour() (see module wx)

See **external documentation**.

setFoldMarginHiColour(This, UseSetting, Fore) -> ok

Types:

This = wxStyledTextCtrl()
UseSetting = boolean()
Fore = wx_colour() (see module wx)

See **external documentation**.

lineDown(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineDownExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineUp(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineUpExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

charLeft(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

charLeftExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

charRight(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

charRightExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordLeft(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordLeftExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordRight(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordRightExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

home(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

homeExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineEnd(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineEndExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

documentStart(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

documentStartExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

documentEnd(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

documentEndExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

pageUp(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

pageUpExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

pageDown(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

pageDownExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

editToggleOvertype(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

cancel(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

deleteBack(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

tab(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

backTab(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

newLine(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

formFeed(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

vCHome(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

vCHomeExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

zoomIn(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

zoomOut(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

delWordLeft(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

delWordRight(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineCut(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineDelete(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineTranspose(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineDuplicate(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lowerCase(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

upperCase(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineScrollDown(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineScrollUp(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

deleteBackNotLine(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

homeDisplay(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

homeDisplayExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineEndDisplay(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineEndDisplayExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

homeWrapExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineEndWrap(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineEndWrapExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

vCHomeWrap(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

vCHomeWrapExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineCopy(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

moveCaretInsideView(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineLength(This, Line) -> integer()

Types:

This = wxStyledTextCtrl()
Line = integer()

See **external documentation**.

braceHighlight(This, Pos1, Pos2) -> ok

Types:

This = wxStyledTextCtrl()
Pos1 = integer()
Pos2 = integer()

See **external documentation**.

braceBadLight(This, Pos) -> ok

Types:

This = wxStyledTextCtrl()
Pos = integer()

See **external documentation**.

braceMatch(This, Pos) -> integer()

Types:

This = wxStyledTextCtrl()
Pos = integer()

See **external documentation**.

getViewEOL(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setViewEOL(This, Visible) -> ok

Types:

This = wxStyledTextCtrl()
Visible = boolean()

See **external documentation**.

setModEventMask(This, Mask) -> ok

Types:

This = wxStyledTextCtrl()

Mask = integer()

See **external documentation**.

getEdgeColumn(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setEdgeColumn(This, Column) -> ok

Types:

This = wxStyledTextCtrl()

Column = integer()

See **external documentation**.

setEdgeMode(This, Mode) -> ok

Types:

This = wxStyledTextCtrl()

Mode = integer()

See **external documentation**.

getEdgeMode(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getEdgeColour(This) -> wx_colour4() (see module wx)

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setEdgeColour(This, EdgeColour) -> ok

Types:

This = wxStyledTextCtrl()

EdgeColour = wx_colour() (see module wx)

See **external documentation**.

searchAnchor(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

searchNext(This, Flags, Text) -> integer()

Types:

```
This = wxStyledTextCtrl()
Flags = integer()
Text = chardata() (see module unicode)
```

See **external documentation**.

searchPrev(This, Flags, Text) -> integer()

Types:

```
This = wxStyledTextCtrl()
Flags = integer()
Text = chardata() (see module unicode)
```

See **external documentation**.

linesOnScreen(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

usePopUp(This, AllowPopUp) -> ok

Types:

```
This = wxStyledTextCtrl()
AllowPopUp = boolean()
```

See **external documentation**.

selectionIsRectangle(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setZoom(This, Zoom) -> ok

Types:

```
This = wxStyledTextCtrl()
Zoom = integer()
```

See **external documentation**.

getZoom(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getModEventMask(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setSTCFocus(This, Focus) -> ok

Types:

```
This = wxStyledTextCtrl()
Focus = boolean()
```

See **external documentation**.

getSTCFocus(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setStatus(This, StatusCode) -> ok

Types:

```
This = wxStyledTextCtrl()
StatusCode = integer()
```

See **external documentation**.

getStatus(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setMouseDownCaptures(This, Captures) -> ok

Types:

```
This = wxStyledTextCtrl()
Captures = boolean()
```

See **external documentation**.

getMouseDownCaptures(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setSTCCursor(This, CursorType) -> ok

Types:

```
This = wxStyledTextCtrl()
CursorType = integer()
```

See **external documentation**.

getSTCCursor(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setControlCharSymbol(This, Symbol) -> ok

Types:

This = wxStyledTextCtrl()

Symbol = integer()

See **external documentation**.

getControlCharSymbol(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordPartLeft(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordPartLeftExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordPartRight(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordPartRightExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setVisiblePolicy(This, VisiblePolicy, VisibleSlop) -> ok

Types:

This = wxStyledTextCtrl()

VisiblePolicy = integer()

VisibleSlop = integer()

See **external documentation**.

delLineLeft(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

delLineRight(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

getXOffset(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

chooseCaretX(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setXCaretPolicy(This, CaretPolicy, CaretSlop) -> ok

Types:

This = wxStyledTextCtrl()
CaretPolicy = integer()
CaretSlop = integer()

See **external documentation**.

setYCaretPolicy(This, CaretPolicy, CaretSlop) -> ok

Types:

This = wxStyledTextCtrl()
CaretPolicy = integer()
CaretSlop = integer()

See **external documentation**.

getPrintWrapMode(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setHotspotActiveForeground(This, UseSetting, Fore) -> ok

Types:

This = wxStyledTextCtrl()
UseSetting = boolean()

```
Fore = wx_colour() (see module wx)
```

See **external documentation**.

setHotspotActiveBackground(This, UseSetting, Back) -> ok

Types:

```
This = wxStyledTextCtrl()
UseSetting = boolean()
Back = wx_colour() (see module wx)
```

See **external documentation**.

setHotspotActiveUnderline(This, Underline) -> ok

Types:

```
This = wxStyledTextCtrl()
Underline = boolean()
```

See **external documentation**.

setHotspotSingleLine(This, SingleLine) -> ok

Types:

```
This = wxStyledTextCtrl()
SingleLine = boolean()
```

See **external documentation**.

paraDownExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

paraUp(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

paraUpExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

positionBefore(This, Pos) -> integer()

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
```

See **external documentation**.

positionAfter(This, Pos) -> integer()

Types:

This = wxStyledTextCtrl()
Pos = integer()

See **external documentation**.

copyRange(This, Start, End) -> ok

Types:

This = wxStyledTextCtrl()
Start = integer()
End = integer()

See **external documentation**.

copyText(This, Length, Text) -> ok

Types:

This = wxStyledTextCtrl()
Length = integer()
Text = chardata() (see module unicode)

See **external documentation**.

setSelectionMode(This, Mode) -> ok

Types:

This = wxStyledTextCtrl()
Mode = integer()

See **external documentation**.

getSelectionMode(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineDownRectExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

lineUpRectExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

charLeftRectExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

charRightRectExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

homeRectExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

vCHomeRectExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

lineEndRectExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

pageUpRectExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

pageDownRectExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

stutteredPageUp(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

stutteredPageUpExtend(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

stutteredPageDown(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

stutteredPageDownExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordLeftEnd(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordLeftEndExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordRightEnd(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

wordRightEndExtend(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setWhitespaceChars(This, Characters) -> ok

Types:

This = wxStyledTextCtrl()

Characters = chardata() (see module unicode)

See **external documentation**.

setCharsDefault(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

autoCompGetCurrent(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

allocate(This, Bytes) -> ok

Types:

```
This = wxStyledTextCtrl()
Bytes = integer()
```

See **external documentation**.

findColumn(This, Line, Column) -> integer()

Types:

```
This = wxStyledTextCtrl()
Line = integer()
Column = integer()
```

See **external documentation**.

getCaretSticky(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setCaretSticky(This, UseCaretStickyBehaviour) -> ok

Types:

```
This = wxStyledTextCtrl()
UseCaretStickyBehaviour = boolean()
```

See **external documentation**.

toggleCaretSticky(This) -> ok

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

setPasteConvertEndings(This, Convert) -> ok

Types:

```
This = wxStyledTextCtrl()
Convert = boolean()
```

See **external documentation**.

getPasteConvertEndings(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

selectionDuplicate(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setCaretLineBackAlpha(This, Alpha) -> ok

Types:

This = wxStyledTextCtrl()
Alpha = integer()

See **external documentation**.

getCaretLineBackAlpha(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

startRecord(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

stopRecord(This) -> ok

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setLexer(This, Lexer) -> ok

Types:

This = wxStyledTextCtrl()
Lexer = integer()

See **external documentation**.

getLexer(This) -> integer()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

colourise(This, Start, End) -> ok

Types:

This = wxStyledTextCtrl()
Start = integer()
End = integer()

See **external documentation**.

setProperty(This, Key, Value) -> ok

Types:

```
This = wxStyledTextCtrl()
Key = chardata() (see module unicode)
Value = chardata() (see module unicode)
```

See **external documentation**.

setKeyWords(This, KeywordSet, KeyWords) -> ok

Types:

```
This = wxStyledTextCtrl()
KeywordSet = integer()
KeyWords = chardata() (see module unicode)
```

See **external documentation**.

setLexerLanguage(This, Language) -> ok

Types:

```
This = wxStyledTextCtrl()
Language = chardata() (see module unicode)
```

See **external documentation**.

getProperty(This, Key) -> charlist() (see module unicode)

Types:

```
This = wxStyledTextCtrl()
Key = chardata() (see module unicode)
```

See **external documentation**.

getStyleBitsNeeded(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getCurrentLine(This) -> integer()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

styleSetSpec(This, StyleNum, Spec) -> ok

Types:

```
This = wxStyledTextCtrl()
StyleNum = integer()
Spec = chardata() (see module unicode)
```

See **external documentation**.

styleSetFont(This, StyleNum, Font) -> ok

Types:

This = wxStyledTextCtrl()
StyleNum = integer()
Font = wxFont() (see module wxFont)

See **external documentation**.

styleSetFontAttr(This, StyleNum, Size, FaceName, Bold, Italic, Underline) -> ok

Types:

This = wxStyledTextCtrl()
StyleNum = integer()
Size = integer()
FaceName = chardata() (see module unicode)
Bold = boolean()
Italic = boolean()
Underline = boolean()

Equivalent to *styleSetFontAttr(This, StyleNum, Size, FaceName, Bold, Italic, Underline, [])*.

styleSetFontAttr(This, StyleNum, Size, FaceName, Bold, Italic, Underline, Option::[Option]) -> ok

Types:

This = wxStyledTextCtrl()
StyleNum = integer()
Size = integer()
FaceName = chardata() (see module unicode)
Bold = boolean()
Italic = boolean()
Underline = boolean()
Option = {encoding, wx_enum() (see module wx)}

See **external documentation**.

Encoding = ?wxFONTENCODING_SYSTEM | ?wxFONTENCODING_DEFAULT | ?wxFONTENCODING_ISO8859_1 | ?wxFONTENCODING_ISO8859_2 | ?wxFONTENCODING_ISO8859_3 | ?wxFONTENCODING_ISO8859_4 | ?wxFONTENCODING_ISO8859_5 | ?wxFONTENCODING_ISO8859_6 | ?wxFONTENCODING_ISO8859_7 | ?wxFONTENCODING_ISO8859_8 | ?wxFONTENCODING_ISO8859_9 | ?wxFONTENCODING_ISO8859_10 | ?wxFONTENCODING_ISO8859_11 | ?wxFONTENCODING_ISO8859_12 | ?wxFONTENCODING_ISO8859_13 | ?wxFONTENCODING_ISO8859_14 | ?wxFONTENCODING_ISO8859_15 | ?wxFONTENCODING_ISO8859_MAX | ?wxFONTENCODING_KOI8 | ?wxFONTENCODING_KOI8_U | ?wxFONTENCODING_ALTERNATIVE | ?wxFONTENCODING_BULGARIAN | ?wxFONTENCODING_CP437 | ?wxFONTENCODING_CP850 | ?wxFONTENCODING_CP852 | ?wxFONTENCODING_CP855 | ?wxFONTENCODING_CP866 | ?wxFONTENCODING_CP874 | ?wxFONTENCODING_CP932 | ?wxFONTENCODING_CP936 | ?wxFONTENCODING_CP949 | ?wxFONTENCODING_CP950 | ?wxFONTENCODING_CP1250 | ?wxFONTENCODING_CP1251 | ?wxFONTENCODING_CP1252 | ?wxFONTENCODING_CP1253 | ?wxFONTENCODING_CP1254 | ?wxFONTENCODING_CP1255 | ?wxFONTENCODING_CP1256 | ?wxFONTENCODING_CP1257 | ?wxFONTENCODING_CP12_MAX | ?wxFONTENCODING_UTF7 | ?wxFONTENCODING_UTF8 | ?

wxFONTENCODING_EUC_JP | ?wxFONTENCODING_UTF16BE | ?wxFONTENCODING_UTF16LE | ?wxFONTENCODING_UTF32BE | ?wxFONTENCODING_UTF32LE | ?wxFONTENCODING_MACROMAN | ?wxFONTENCODING_MACJAPANESE | ?wxFONTENCODING_MACCHINESETRAD | ?wxFONTENCODING_MACKOREAN | ?wxFONTENCODING_MACARABIC | ?wxFONTENCODING_MACHEBREW | ?wxFONTENCODING_MACGREEK | ?wxFONTENCODING_MACCYRILLIC | ?wxFONTENCODING_MACDEVANAGARI | ?wxFONTENCODING_MACGURMUKHI | ?wxFONTENCODING_MACGUJARATI | ?wxFONTENCODING_MACORIYA | ?wxFONTENCODING_MACBENGALI | ?wxFONTENCODING_MACTAMIL | ?wxFONTENCODING_MACTELUGU | ?wxFONTENCODING_MACKANNADA | ?wxFONTENCODING_MACMALAJALAM | ?wxFONTENCODING_MACSINHALESE | ?wxFONTENCODING_MACBURMESE | ?wxFONTENCODING_MACKHMER | ?wxFONTENCODING_MACTHAI | ?wxFONTENCODING_MACLAOTIAN | ?wxFONTENCODING_MACGEORGIAN | ?wxFONTENCODING_MACARMENIAN | ?wxFONTENCODING_MACCHINESESIMP | ?wxFONTENCODING_MACTIBETAN | ?wxFONTENCODING_MACMONGOLIAN | ?wxFONTENCODING_MACETHIOPIC | ?wxFONTENCODING_MACCENTRALEUR | ?wxFONTENCODING_MACVIATNAMESE | ?wxFONTENCODING_MACARABICEXT | ?wxFONTENCODING_MACSYMBOL | ?wxFONTENCODING_MACDINGBATS | ?wxFONTENCODING_MACTURKISH | ?wxFONTENCODING_MACCROATIAN | ?wxFONTENCODING_MACICELANDIC | ?wxFONTENCODING_MACROMANIAN | ?wxFONTENCODING_MACCELTIC | ?wxFONTENCODING_MACGAELIC | ?wxFONTENCODING_MACKEYBOARD | ?wxFONTENCODING_MAX | ?wxFONTENCODING_MACMIN | ?wxFONTENCODING_MACMAX | ?wxFONTENCODING_UTF16 | ?wxFONTENCODING_UTF32 | ?wxFONTENCODING_UNICODE | ?wxFONTENCODING_GB2312 | ?wxFONTENCODING_BIG5 | ?wxFONTENCODING_SHIFT_JIS

`styleSetCharacterSet(This, Style, CharacterSet) -> ok`

Types:

`This = wxStyledTextCtrl()`
`Style = integer()`
`CharacterSet = integer()`

See **external documentation**.

`styleSetFontEncoding(This, Style, Encoding) -> ok`

Types:

`This = wxStyledTextCtrl()`
`Style = integer()`
`Encoding = wx_enum() (see module wx)`

See **external documentation**.

Encoding = ?wxFONTENCODING_SYSTEM | ?wxFONTENCODING_DEFAULT | ?wxFONTENCODING_ISO8859_1 | ?wxFONTENCODING_ISO8859_2 | ?wxFONTENCODING_ISO8859_3 | ?wxFONTENCODING_ISO8859_4 | ?wxFONTENCODING_ISO8859_5 | ?wxFONTENCODING_ISO8859_6 | ?wxFONTENCODING_ISO8859_7 | ?wxFONTENCODING_ISO8859_8 | ?wxFONTENCODING_ISO8859_9 | ?wxFONTENCODING_ISO8859_10 | ?wxFONTENCODING_ISO8859_11 | ?wxFONTENCODING_ISO8859_12 | ?wxFONTENCODING_ISO8859_13 | ?wxFONTENCODING_ISO8859_14 | ?wxFONTENCODING_ISO8859_15 | ?wxFONTENCODING_ISO8859_MAX | ?wxFONTENCODING_KOI8 | ?wxFONTENCODING_KOI8_U | ?wxFONTENCODING_ALTERNATIVE | ?wxFONTENCODING_BULGARIAN | ?wxFONTENCODING_CP437 | ?wxFONTENCODING_CP850 | ?wxFONTENCODING_CP852 | ?wxFONTENCODING_CP855 | ?wxFONTENCODING_CP866 | ?wxFONTENCODING_CP874 | ?

wxFONTENCODING_CP932 | ?wxFONTENCODING_CP936 | ?wxFONTENCODING_CP949 | ?wxFONTENCODING_CP950 | ?wxFONTENCODING_CP1250 | ?wxFONTENCODING_CP1251 | ?wxFONTENCODING_CP1252 | ?wxFONTENCODING_CP1253 | ?wxFONTENCODING_CP1254 | ?wxFONTENCODING_CP1255 | ?wxFONTENCODING_CP1256 | ?wxFONTENCODING_CP1257 | ?wxFONTENCODING_CP12_MAX | ?wxFONTENCODING_UTF7 | ?wxFONTENCODING_UTF8 | ?wxFONTENCODING_EUC_JP | ?wxFONTENCODING_UTF16BE | ?wxFONTENCODING_UTF16LE | ?wxFONTENCODING_UTF32BE | ?wxFONTENCODING_UTF32LE | ?wxFONTENCODING_MACROMAN | ?wxFONTENCODING_MACJAPANESE | ?wxFONTENCODING_MACCHINESETRAD | ?wxFONTENCODING_MACKOREAN | ?wxFONTENCODING_MACARABIC | ?wxFONTENCODING_MACHEBREW | ?wxFONTENCODING_MACGREEK | ?wxFONTENCODING_MACCYRILLIC | ?wxFONTENCODING_MACDEVANAGARI | ?wxFONTENCODING_MACGURMUKHI | ?wxFONTENCODING_MACGUJARATI | ?wxFONTENCODING_MACORIYA | ?wxFONTENCODING_MACBENGALI | ?wxFONTENCODING_MACTAMIL | ?wxFONTENCODING_MACTELUGU | ?wxFONTENCODING_MACKANNADA | ?wxFONTENCODING_MACMALAJALAM | ?wxFONTENCODING_MACSINHALESE | ?wxFONTENCODING_MACBURMESE | ?wxFONTENCODING_MACKHMER | ?wxFONTENCODING_MACTHAI | ?wxFONTENCODING_MACLAOTIAN | ?wxFONTENCODING_MACGEORGIAN | ?wxFONTENCODING_MACARMENIAN | ?wxFONTENCODING_MACCHINESESIMP | ?wxFONTENCODING_MACTIBETAN | ?wxFONTENCODING_MACMONGOLIAN | ?wxFONTENCODING_MACETHIOPIC | ?wxFONTENCODING_MACCENTRALEUR | ?wxFONTENCODING_MACVIATNAMESE | ?wxFONTENCODING_MACARABICEXT | ?wxFONTENCODING_MACSYMBOL | ?wxFONTENCODING_MACDINGBATS | ?wxFONTENCODING_MACTURKISH | ?wxFONTENCODING_MACCROATIAN | ?wxFONTENCODING_MACICELANDIC | ?wxFONTENCODING_MACROMANIAN | ?wxFONTENCODING_MACCELTIC | ?wxFONTENCODING_MACGAELIC | ?wxFONTENCODING_MACKEYBOARD | ?wxFONTENCODING_MAX | ?wxFONTENCODING_MACMIN | ?wxFONTENCODING_MACMAX | ?wxFONTENCODING_UTF16 | ?wxFONTENCODING_UTF32 | ?wxFONTENCODING_UNICODE | ?wxFONTENCODING_GB2312 | ?wxFONTENCODING_BIG5 | ?wxFONTENCODING_SHIFT_JIS

`cmdKeyExecute(This, Cmd) -> ok`

Types:

`This = wxStyledTextCtrl()`
`Cmd = integer()`

See **external documentation**.

`setMargins(This, Left, Right) -> ok`

Types:

`This = wxStyledTextCtrl()`
`Left = integer()`
`Right = integer()`

See **external documentation**.

`getSelection(This) -> {StartPos::integer(), EndPos::integer()}`

Types:

`This = wxStyledTextCtrl()`

See **external documentation**.

pointFromPosition(This, Pos) -> {X::integer(), Y::integer()}

Types:

This = wxStyledTextCtrl()

Pos = integer()

See **external documentation**.

scrollToLine(This, Line) -> ok

Types:

This = wxStyledTextCtrl()

Line = integer()

See **external documentation**.

scrollToColumn(This, Column) -> ok

Types:

This = wxStyledTextCtrl()

Column = integer()

See **external documentation**.

sendMsg(This, Msg) -> integer()

Types:

This = wxStyledTextCtrl()

Msg = integer()

Equivalent to *sendMsg(This, Msg, [])*.

sendMsg(This, Msg, Option::[Option]) -> integer()

Types:

This = wxStyledTextCtrl()

Msg = integer()

Option = {wp, integer()} | {lp, integer()}

See **external documentation**.

setVScrollBar(This, Bar) -> ok

Types:

This = wxStyledTextCtrl()

Bar = wxScrollBar() (see module wxScrollBar)

See **external documentation**.

setHScrollBar(This, Bar) -> ok

Types:

This = wxStyledTextCtrl()

Bar = wxScrollBar() (see module wxScrollBar)

See **external documentation**.

getLastKeydownProcessed(This) -> boolean()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

setLastKeydownProcessed(This, Val) -> ok

Types:

This = wxStyledTextCtrl()
Val = boolean()

See **external documentation**.

saveFile(This, Filename) -> boolean()

Types:

This = wxStyledTextCtrl()
Filename = chardata() (see module unicode)

See **external documentation**.

loadFile(This, Filename) -> boolean()

Types:

This = wxStyledTextCtrl()
Filename = chardata() (see module unicode)

See **external documentation**.

doDragOver(This, X, Y, Def) -> wx_enum() (see module wx)

Types:

This = wxStyledTextCtrl()
X = integer()
Y = integer()
Def = wx_enum() (see module wx)

See **external documentation**.

Def = ?wxDragError | ?wxDragNone | ?wxDragCopy | ?wxDragMove | ?wxDragLink | ?wxDragCancel
Res = ?wxDragError | ?wxDragNone | ?wxDragCopy | ?wxDragMove | ?wxDragLink | ?wxDragCancel

doDropText(This, X, Y, Data) -> boolean()

Types:

This = wxStyledTextCtrl()
X = integer()
Y = integer()
Data = chardata() (see module unicode)

See **external documentation**.

getUseAntiAliasing(This) -> boolean()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

addTextRaw(This, Text) -> ok

Types:

```
This = wxStyledTextCtrl()
Text = binary()
```

See **external documentation**.

insertTextRaw(This, Pos, Text) -> ok

Types:

```
This = wxStyledTextCtrl()
Pos = integer()
Text = binary()
```

See **external documentation**.

getCurLineRaw(This) -> Result

Types:

```
Result = {Res::binary(), LinePos::integer()}
This = wxStyledTextCtrl()
```

See **external documentation**.

getLineRaw(This, Line) -> binary()

Types:

```
This = wxStyledTextCtrl()
Line = integer()
```

See **external documentation**.

getSelectedTextRaw(This) -> binary()

Types:

```
This = wxStyledTextCtrl()
```

See **external documentation**.

getTextRangeRaw(This, StartPos, EndPos) -> binary()

Types:

```
This = wxStyledTextCtrl()
StartPos = integer()
EndPos = integer()
```

See **external documentation**.

setTextRaw(This, Text) -> ok

Types:

```
This = wxStyledTextCtrl()
```

Text = binary()

See **external documentation**.

getTextRaw(This) -> binary()

Types:

This = wxStyledTextCtrl()

See **external documentation**.

appendTextRaw(This, Text) -> ok

Types:

This = wxStyledTextCtrl()

Text = binary()

See **external documentation**.

destroy(This::wxStyledTextCtrl()) -> ok

Destroys this object, do not use object again

wxStyledTextEvent

Erlang module

See external documentation: **wxStyledTextEvent**.

Use *wxEvtHandler:connect/3* with EventType:

stc_change, *stc_styleneeded*, *stc_charadded*, *stc_savepointreached*, *stc_savepointleft*, *stc_romodifyattempt*, *stc_key*, *stc_doubleclick*, *stc_updateui*, *stc_modified*, *stc_macrorecord*, *stc_marginclick*, *stc_needshown*, *stc_painted*, *stc_userlistselection*, *stc_uridropped*, *stc_dwellstart*, *stc_dwellend*, *stc_start_drag*, *stc_drag_over*, *stc_do_drop*, *stc_zoom*, *stc_hotspot_click*, *stc_hotspot_dclick*, *stc_calltip_click*, *stc_autocomp_selection*

See also the message variant *#wxStyledText{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxStyledTextEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getPosition(This) -> integer()`

Types:

`This = wxStyledTextEvent()`

See **external documentation**.

`getKey(This) -> integer()`

Types:

`This = wxStyledTextEvent()`

See **external documentation**.

`getModifiers(This) -> integer()`

Types:

`This = wxStyledTextEvent()`

See **external documentation**.

`getModificationType(This) -> integer()`

Types:

`This = wxStyledTextEvent()`

See **external documentation**.

getText(This) -> charlist() (see module unicode)

Types:

This = wxStyledTextEvent()

See **external documentation**.

getLength(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getLinesAdded(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getLine(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getFoldLevelNow(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getFoldLevelPrev(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getMargin(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getMessage(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getWParam(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getLParam(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getListType(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getX(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getY(This) -> integer()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getDragText(This) -> charlist() (see module unicode)

Types:

This = wxStyledTextEvent()

See **external documentation**.

getDragAllowMove(This) -> boolean()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getDragResult(This) -> wx_enum() (see module wx)

Types:

This = wxStyledTextEvent()

See **external documentation**.

Res = ?wxDragError | ?wxDragNone | ?wxDragCopy | ?wxDragMove | ?wxDragLink | ?wxDragCancel

getShift(This) -> boolean()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getControl(This) -> boolean()

Types:

This = wxStyledTextEvent()

See **external documentation**.

getAlt(This) -> boolean()

Types:

This = wxStyledTextEvent()

See **external documentation**.

wxSysColourChangedEvent

Erlang module

See external documentation: **wxSysColourChangedEvent**.

Use *wxEvtHandler:connect/3* with EventType:

sys_colour_changed

See also the message variant *#wxSysColourChanged{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxSysColourChangedEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxSystemOptions

Erlang module

See external documentation: **wxSystemOptions**.

DATA TYPES

`wxSystemOptions()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getOption(Name) -> charlist() (see module unicode)`

Types:

`Name = chardata() (see module unicode)`

See **external documentation**.

`getOptionInt(Name) -> integer()`

Types:

`Name = chardata() (see module unicode)`

See **external documentation**.

`hasOption(Name) -> boolean()`

Types:

`Name = chardata() (see module unicode)`

See **external documentation**.

`isFalse(Name) -> boolean()`

Types:

`Name = chardata() (see module unicode)`

See **external documentation**.

`setOption(Name, Value) -> ok`

Types:

`Name = chardata() (see module unicode)`

`Value = integer()`

See **external documentation**.
Also:
setOption(Name, Value) -> ok when
Name::unicode:chardata(), Value::unicode:chardata().

wxSystemSettings

Erlang module

See external documentation: **wxSystemSettings**.

DATA TYPES

`wxSystemSettings()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getColour(Index) -> wx_colour4() (see module wx)`

Types:

`Index = wx_enum() (see module wx)`

See **external documentation**.
Index = ?wxSYS_COLOUR_SCROLLBAR | ?wxSYS_COLOUR_BACKGROUND | ?wxSYS_COLOUR_DESKTOP | ?wxSYS_COLOUR_ACTIVECAPTION | ?wxSYS_COLOUR_INACTIVECAPTION | ?wxSYS_COLOUR_MENU | ?wxSYS_COLOUR_WINDOW | ?wxSYS_COLOUR_WINDOWFRAME | ?wxSYS_COLOUR_MENUTEXT | ?wxSYS_COLOUR_WINDOWTEXT | ?wxSYS_COLOUR_CAPTIONTEXT | ?wxSYS_COLOUR_ACTIVEBORDER | ?wxSYS_COLOUR_INACTIVEBORDER | ?wxSYS_COLOUR_APPWORKSPACE | ?wxSYS_COLOUR_HIGHLIGHT | ?wxSYS_COLOUR_HIGHLIGHTTEXT | ?wxSYS_COLOUR_BTNFACE | ?wxSYS_COLOUR_3DFACE | ?wxSYS_COLOUR_BTNSHADOW | ?wxSYS_COLOUR_3DSHADOW | ?wxSYS_COLOUR_GRAYTEXT | ?wxSYS_COLOUR_BTNTEXT | ?wxSYS_COLOUR_INACTIVECAPTIONTEXT | ?wxSYS_COLOUR_BTNHIGHLIGHT | ?wxSYS_COLOUR_BTNHILIGHT | ?wxSYS_COLOUR_3DHIGHLIGHT | ?wxSYS_COLOUR_3DHILIGHT | ?wxSYS_COLOUR_3DDKSHADOW | ?wxSYS_COLOUR_3DLIGHT | ?wxSYS_COLOUR_INFOTEXT | ?wxSYS_COLOUR_INFOBK | ?wxSYS_COLOUR_LISTBOX | ?wxSYS_COLOUR_HOTLIGHT | ?wxSYS_COLOUR_GRADIENTACTIVECAPTION | ?wxSYS_COLOUR_GRADIENTINACTIVECAPTION | ?wxSYS_COLOUR_MENUHILIGHT | ?wxSYS_COLOUR_MENUBAR | ?wxSYS_COLOUR_LISTBOXTEXT | ?wxSYS_COLOUR_MAX

`getFont(Index) -> wxFont() (see module wxFont)`

Types:

`Index = wx_enum() (see module wx)`

See **external documentation**.
Index = ?wxSYS_OEM_FIXED_FONT | ?wxSYS_ANSI_FIXED_FONT | ?wxSYS_ANSI_VAR_FONT | ?wxSYS_SYSTEM_FONT | ?wxSYS_DEVICE_DEFAULT_FONT | ?wxSYS_DEFAULT_PALETTE | ?wxSYS_SYSTEM_FIXED_FONT | ?wxSYS_DEFAULT_GUI_FONT | ?wxSYS_ICONTITLE_FONT

`getMetric(Index) -> integer()`

Types:

`Index = wx_enum() (see module wx)`

Equivalent to *getMetric(Index, [])*.

getMetric(Index, Option::[Option]) -> integer()

Types:

Index = wx_enum() (see module wx)

Option = {win, wxWindow() (see module wxWindow)}

See **external documentation**.
Index = ?wxSYS_MOUSE_BUTTONS | ?wxSYS_BORDER_X | ?wxSYS_BORDER_Y | ?wxSYS_CURSOR_X | ?wxSYS_CURSOR_Y | ?wxSYS_DCLICK_X | ?wxSYS_DCLICK_Y | ?wxSYS_DRAG_X | ?wxSYS_DRAG_Y | ?wxSYS_EDGE_X | ?wxSYS_EDGE_Y | ?wxSYS_HSCROLL_ARROW_X | ?wxSYS_HSCROLL_ARROW_Y | ?wxSYS_HTHUMB_X | ?wxSYS_ICON_X | ?wxSYS_ICON_Y | ?wxSYS_ICONSPACING_X | ?wxSYS_ICONSPACING_Y | ?wxSYS_WINDOWMIN_X | ?wxSYS_WINDOWMIN_Y | ?wxSYS_SCREEN_X | ?wxSYS_SCREEN_Y | ?wxSYS_FRAMESIZE_X | ?wxSYS_FRAMESIZE_Y | ?wxSYS_SMALLICON_X | ?wxSYS_SMALLICON_Y | ?wxSYS_HSCROLL_Y | ?wxSYS_VSCROLL_X | ?wxSYS_VSCROLL_ARROW_X | ?wxSYS_VSCROLL_ARROW_Y | ?wxSYS_VTHUMB_Y | ?wxSYS_CAPTION_Y | ?wxSYS_MENU_Y | ?wxSYS_NETWORK_PRESENT | ?wxSYS_PENWINDOWS_PRESENT | ?wxSYS_SHOW_SOUNDS | ?wxSYS_SWAP_BUTTONS

getScreenType() -> wx_enum() (see module wx)

See **external documentation**.
Res = ?wxSYS_SCREEN_NONE | ?wxSYS_SCREEN_TINY | ?wxSYS_SCREEN_PDA | ?wxSYS_SCREEN_SMALL | ?wxSYS_SCREEN_DESKTOP

wxTaskBarIcon

Erlang module

See external documentation: **wxTaskBarIcon**.

This class is derived (and can use functions) from:
wxEvtHandler

DATA TYPES

`wxTaskBarIcon()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxTaskBarIcon()`

See **external documentation**.

`popupMenu(This, Menu) -> boolean()`

Types:

```
This = wxTaskBarIcon()
Menu = wxMenu() (see module wxMenu)
```

See **external documentation**.

`removeIcon(This) -> boolean()`

Types:

```
This = wxTaskBarIcon()
```

See **external documentation**.

`setIcon(This, Icon) -> boolean()`

Types:

```
This = wxTaskBarIcon()
Icon = wxIcon() (see module wxIcon)
```

Equivalent to *setIcon(This, Icon, [])*.

`setIcon(This, Icon, Option::[Option]) -> boolean()`

Types:

```
This = wxTaskBarIcon()
Icon = wxIcon() (see module wxIcon)
Option = {tooltip, chardata() (see module unicode)}
```

See **external documentation**.

destroy(This::wxTaskBarIcon()) -> ok

Destroys this object, do not use object again

wxTaskBarIconEvent

Erlang module

See external documentation: **wxTaskBarIconEvent**.

Use *wxEvtHandler:connect/3* with EventType:

taskbar_move, taskbar_left_down, taskbar_left_up, taskbar_right_down, taskbar_right_up, taskbar_left_dclick, taskbar_right_dclick

See also the message variant *#wxTaskBarIcon{}* event record type.

This class is derived (and can use functions) from:
wxEvent

DATA TYPES

`wxTaskBarIconEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxTextAttr

Erlang module

See external documentation: **wxTextAttr**.

DATA TYPES

`wxTextAttr()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxTextAttr()`

See **external documentation**.

`new(ColText) -> wxTextAttr()`

Types:

`ColText = wx_colour() (see module wx)`

Equivalent to *new(ColText, [])*.

`new(ColText, Option::[Option]) -> wxTextAttr()`

Types:

`ColText = wx_colour() (see module wx)`

`Option = {colBack, wx_colour() (see module wx)} | {font, wxFont() (see module wxFont)} | {alignment, wx_enum() (see module wx)}`

See **external documentation**.
Alignment = ?wxTEXT_ALIGNMENT_DEFAULT | ?wxTEXT_ALIGNMENT_LEFT | ?wxTEXT_ALIGNMENT_CENTRE | ?wxTEXT_ALIGNMENT_CENTER | ?wxTEXT_ALIGNMENT_RIGHT | ?wxTEXT_ALIGNMENT_JUSTIFIED

`getAlignment(This) -> wx_enum() (see module wx)`

Types:

`This = wxTextAttr()`

See **external documentation**.
Res = ?wxTEXT_ALIGNMENT_DEFAULT | ?wxTEXT_ALIGNMENT_LEFT | ?wxTEXT_ALIGNMENT_CENTRE | ?wxTEXT_ALIGNMENT_CENTER | ?wxTEXT_ALIGNMENT_RIGHT | ?wxTEXT_ALIGNMENT_JUSTIFIED

`getBackgroundColour(This) -> wx_colour4() (see module wx)`

Types:

`This = wxTextAttr()`

See **external documentation**.

getFont(This) -> wxFont() (see module wxFont)

Types:

This = wxTextAttr()

See **external documentation**.

getLeftIndent(This) -> integer()

Types:

This = wxTextAttr()

See **external documentation**.

getLeftSubIndent(This) -> integer()

Types:

This = wxTextAttr()

See **external documentation**.

getRightIndent(This) -> integer()

Types:

This = wxTextAttr()

See **external documentation**.

getTabs(This) -> [integer()]

Types:

This = wxTextAttr()

See **external documentation**.

getTextColour(This) -> wx_colour4() (see module wx)

Types:

This = wxTextAttr()

See **external documentation**.

hasBackgroundColour(This) -> boolean()

Types:

This = wxTextAttr()

See **external documentation**.

hasFont(This) -> boolean()

Types:

This = wxTextAttr()

See **external documentation**.

hasTextColour(This) -> boolean()

Types:

This = wxTextAttr()

See **external documentation**.

getFlags(This) -> integer()

Types:

This = wxTextAttr()

See **external documentation**.

isDefault(This) -> boolean()

Types:

This = wxTextAttr()

See **external documentation**.

setAlignment(This, Alignment) -> ok

Types:

This = wxTextAttr()

Alignment = wx_enum() (see module wx)

See **external documentation**.

Alignment = ?wxTEXT_ALIGNMENT_DEFAULT | ?wxTEXT_ALIGNMENT_LEFT | ?wxTEXT_ALIGNMENT_CENTRE | ?wxTEXT_ALIGNMENT_CENTER | ?wxTEXT_ALIGNMENT_RIGHT | ?wxTEXT_ALIGNMENT_JUSTIFIED

setBackgroundColour(This, ColBack) -> ok

Types:

This = wxTextAttr()

ColBack = wx_colour() (see module wx)

See **external documentation**.

setFlags(This, Flags) -> ok

Types:

This = wxTextAttr()

Flags = integer()

See **external documentation**.

setFont(This, Font) -> ok

Types:

This = wxTextAttr()

Font = wxFont() (see module wxFont)

Equivalent to *setFont(This, Font, [])*.

setFont(This, Font, Option::[Option]) -> ok

Types:

This = wxTextAttr()

Font = wxFont() (see module wxFont)

Option = {flags, integer()}

See **external documentation**.

setLeftIndent(This, Indent) -> ok

Types:

```
This = wxTextAttr()
Indent = integer()
```

Equivalent to *setLeftIndent(This, Indent, [])*.

setLeftIndent(This, Indent, Option::[Option]) -> ok

Types:

```
This = wxTextAttr()
Indent = integer()
Option = {subIndent, integer()}
```

See **external documentation**.

setRightIndent(This, Indent) -> ok

Types:

```
This = wxTextAttr()
Indent = integer()
```

See **external documentation**.

setTabs(This, Tabs) -> ok

Types:

```
This = wxTextAttr()
Tabs = [integer()]
```

See **external documentation**.

setTextColour(This, ColText) -> ok

Types:

```
This = wxTextAttr()
ColText = wx_colour() (see module wx)
```

See **external documentation**.

destroy(This::wxTextAttr()) -> ok

Destroys this object, do not use object again

wxTextCtrl

Erlang module

See external documentation: **wxTextCtrl**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxTextCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxTextCtrl()`

See **external documentation**.

`new(Parent, Id) -> wxTextCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxTextCtrl()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {value, chardata() (see module unicode)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
| {validator, wx_object() (see module wx)}
```

See **external documentation**.

`appendText(This, Text) -> ok`

Types:

```
This = wxTextCtrl()
Text = chardata() (see module unicode)
```

See **external documentation**.

`canCopy(This) -> boolean()`

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

canCut(This) -> boolean()

Types:

This = wxTextCtrl()

See **external documentation**.

canPaste(This) -> boolean()

Types:

This = wxTextCtrl()

See **external documentation**.

canRedo(This) -> boolean()

Types:

This = wxTextCtrl()

See **external documentation**.

canUndo(This) -> boolean()

Types:

This = wxTextCtrl()

See **external documentation**.

clear(This) -> ok

Types:

This = wxTextCtrl()

See **external documentation**.

copy(This) -> ok

Types:

This = wxTextCtrl()

See **external documentation**.

create(This, Parent, Id) -> boolean()

Types:

This = wxTextCtrl()

Parent = wxWindow() (see module wxWindow)

Id = integer()

Equivalent to *create(This, Parent, Id, [])*.

create(This, Parent, Id, Option::[Option]) -> boolean()

Types:

This = wxTextCtrl()

Parent = wxWindow() (see module wxWindow)

```
Id = integer()
Option = {value, chardata() (see module unicode)} | {pos, {X::integer(),
Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()}
| {validator, wx_object() (see module wx)}
```

See **external documentation**.

cut(This) -> ok

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

discardEdits(This) -> ok

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

emulateKeyPress(This, Event) -> boolean()

Types:

```
This = wxTextCtrl()
Event = wxKeyEvent() (see module wxKeyEvent)
```

See **external documentation**.

getDefaultStyle(This) -> wxTextAttr() (see module wxTextAttr)

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

getInsertionPoint(This) -> integer()

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

getLastPosition(This) -> integer()

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

getLineLength(This, LineNo) -> integer()

Types:

```
This = wxTextCtrl()
LineNo = integer()
```

See **external documentation**.

getLineText(This, LineNo) -> charlist() (see module unicode)

Types:

This = wxTextCtrl()

LineNo = integer()

See **external documentation**.

getNumberOfLines(This) -> integer()

Types:

This = wxTextCtrl()

See **external documentation**.

getRange(This, From, To) -> charlist() (see module unicode)

Types:

This = wxTextCtrl()

From = integer()

To = integer()

See **external documentation**.

getSelection(This) -> {From::integer(), To::integer()}

Types:

This = wxTextCtrl()

See **external documentation**.

getStringSelection(This) -> charlist() (see module unicode)

Types:

This = wxTextCtrl()

See **external documentation**.

getStyle(This, Position, Style) -> boolean()

Types:

This = wxTextCtrl()

Position = integer()

Style = wxTextAttr() (see module wxTextAttr)

See **external documentation**.

getValue(This) -> charlist() (see module unicode)

Types:

This = wxTextCtrl()

See **external documentation**.

isEditable(This) -> boolean()

Types:

This = wxTextCtrl()

See **external documentation**.

isModified(This) -> boolean()

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

isMultiLine(This) -> boolean()

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

isSingleLine(This) -> boolean()

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

loadFile(This, File) -> boolean()

Types:

```
This = wxTextCtrl()
File = chardata() (see module unicode)
```

Equivalent to *loadFile(This, File, [])*.

loadFile(This, File, Option::[Option]) -> boolean()

Types:

```
This = wxTextCtrl()
File = chardata() (see module unicode)
Option = {fileType, integer()}
```

See **external documentation**.

markDirty(This) -> ok

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

paste(This) -> ok

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

positionToXY(This, Pos) -> Result

Types:

```
Result = {Res::boolean(), X::integer(), Y::integer()}
```

```
This = wxTextCtrl()
Pos = integer()
```

See **external documentation**.

redo(This) -> ok

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

remove(This, From, To) -> ok

Types:

```
This = wxTextCtrl()
From = integer()
To = integer()
```

See **external documentation**.

replace(This, From, To, Value) -> ok

Types:

```
This = wxTextCtrl()
From = integer()
To = integer()
Value = chardata() (see module unicode)
```

See **external documentation**.

saveFile(This) -> boolean()

Types:

```
This = wxTextCtrl()
```

Equivalent to *saveFile(This, [])*.

saveFile(This, Option::[Option]) -> boolean()

Types:

```
This = wxTextCtrl()
Option = {file, chardata() (see module unicode)} | {fileType, integer()}
```

See **external documentation**.

setDefaultStyle(This, Style) -> boolean()

Types:

```
This = wxTextCtrl()
Style = wxTextAttr() (see module wxTextAttr)
```

See **external documentation**.

setEditable(This, Editable) -> ok

Types:

```
This = wxTextCtrl()
Editable = boolean()
```

See **external documentation**.

setInsertionPoint(This, Pos) -> ok

Types:

```
This = wxTextCtrl()
Pos = integer()
```

See **external documentation**.

setInsertionPointEnd(This) -> ok

Types:

```
This = wxTextCtrl()
```

See **external documentation**.

setMaxLength(This, Len) -> ok

Types:

```
This = wxTextCtrl()
Len = integer()
```

See **external documentation**.

setSelection(This, From, To) -> ok

Types:

```
This = wxTextCtrl()
From = integer()
To = integer()
```

See **external documentation**.

setStyle(This, Start, End, Style) -> boolean()

Types:

```
This = wxTextCtrl()
Start = integer()
End = integer()
Style = wxTextAttr() (see module wxTextAttr)
```

See **external documentation**.

setValue(This, Value) -> ok

Types:

```
This = wxTextCtrl()
Value = chardata() (see module unicode)
```

See **external documentation**.

showPosition(This, Pos) -> ok

Types:

This = wxTextCtrl()

Pos = integer()

See **external documentation**.

undo(This) -> ok

Types:

This = wxTextCtrl()

See **external documentation**.

writeText(This, Text) -> ok

Types:

This = wxTextCtrl()

Text = chardata() (see module unicode)

See **external documentation**.

xYToPosition(This, X, Y) -> integer()

Types:

This = wxTextCtrl()

X = integer()

Y = integer()

See **external documentation**.

destroy(This::wxTextCtrl()) -> ok

Destroys this object, do not use object again

wxTextDataObject

Erlang module

See external documentation: **wxTextDataObject**.

This class is derived (and can use functions) from:
wxDataObject

DATA TYPES

`wxTextDataObject()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxTextDataObject()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxTextDataObject()`

Types:

```
Option = {text, chardata() (see module unicode)}
```

See **external documentation**.

`getTextLength(This) -> integer()`

Types:

```
This = wxTextDataObject()
```

See **external documentation**.

`getText(This) -> charlist() (see module unicode)`

Types:

```
This = wxTextDataObject()
```

See **external documentation**.

`setText(This, Text) -> ok`

Types:

```
This = wxTextDataObject()
Text = chardata() (see module unicode)
```

See **external documentation**.

`destroy(This::wxTextDataObject()) -> ok`

Destroys this object, do not use object again

wxTextEntryDialog

Erlang module

See external documentation: **wxTextEntryDialog**.

This class is derived (and can use functions) from:
wxDialog
wxTopLevelWindow
wxWindow
wxEvtHandler

DATA TYPES

`wxTextEntryDialog()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new(Parent, Message) -> wxTextEntryDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`
`Message = chardata() (see module unicode)`

Equivalent to *new(Parent, Message, [])*.

`new(Parent, Message, Option::[Option]) -> wxTextEntryDialog()`

Types:

`Parent = wxWindow() (see module wxWindow)`
`Message = chardata() (see module unicode)`
`Option = {caption, chardata() (see module unicode)} | {value, chardata() (see module unicode)} | {style, integer()} | {pos, {X::integer(), Y::integer()}}`

See **external documentation**.

`getValue(This) -> charlist() (see module unicode)`

Types:

`This = wxTextEntryDialog()`

See **external documentation**.

`setValue(This, Val) -> ok`

Types:

`This = wxTextEntryDialog()`
`Val = chardata() (see module unicode)`

See **external documentation**.

destroy(This::wxTextEntryDialog()) -> ok

Destroys this object, do not use object again

wxToggleButton

Erlang module

See external documentation: **wxToggleButton**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxToggleButton()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxToggleButton()`

See **external documentation**.

`new(Parent, Id, Label) -> wxToggleButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *new(Parent, Id, Label, [])*.

`new(Parent, Id, Label, Option::[Option]) -> wxToggleButton()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

`create(This, Parent, Id, Label) -> boolean()`

Types:

```
This = wxToggleButton()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
```

Equivalent to *create(This, Parent, Id, Label, [])*.

```
create(This, Parent, Id, Label, Option::[Option]) -> boolean()
```

Types:

```
This = wxToggleButton()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Label = chardata() (see module unicode)
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()} | {validator, wx_object() (see module
wx)}
```

See **external documentation**.

```
getValue(This) -> boolean()
```

Types:

```
This = wxToggleButton()
```

See **external documentation**.

```
setValue(This, State) -> ok
```

Types:

```
This = wxToggleButton()
State = boolean()
```

See **external documentation**.

```
destroy(This::wxToggleButton()) -> ok
```

Destroys this object, do not use object again

wxToolBar

Erlang module

See external documentation: **wxToolBar**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxToolBar()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`addControl(This, Control) -> wx_object() (see module wx)`

Types:

```
This = wxToolBar()
Control = wxControl() (see module wxControl)
```

See **external documentation**.

`addSeparator(This) -> wx_object() (see module wx)`

Types:

```
This = wxToolBar()
```

See **external documentation**.

`addTool(This, Tool) -> wx_object() (see module wx)`

Types:

```
This = wxToolBar()
Tool = wx_object() (see module wx)
```

See **external documentation**.

`addTool(This, Toolid, Bitmap) -> wx_object() (see module wx)`

Types:

```
This = wxToolBar()
Toolid = integer()
Bitmap = wxBitmap() (see module wxBitmap)
```

Equivalent to *addTool(This, Toolid, Bitmap, [])*.

`addTool(This, Toolid, Label, Bitmap) -> wx_object() (see module wx)`

Types:

This = wxToolBar()
Toolid = integer()
Label = chardata() (see module unicode)
Bitmap = wxBitmap() (see module wxBitmap)

See **external documentation**.
Also:
addTool(This, Toolid, Bitmap, BmpDisabled) -> wx:wx_object() when
This::wxToolBar(), Toolid::integer(), Bitmap::wxBitmap:wxBitmap(), BmpDisabled::wxBitmap:wxBitmap();
(This, Toolid, Bitmap, [Option]) -> wx:wx_object() when
This::wxToolBar(), Toolid::integer(), Bitmap::wxBitmap:wxBitmap(),
Option :: {shortHelpString, unicode:chardata()}
| {longHelpString, unicode:chardata()}.

Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

addTool(This, Toolid, Label, Bitmap, BmpDisabled) -> wx_object() (see module wx)

Types:

This = wxToolBar()
Toolid = integer()
Label = chardata() (see module unicode)
Bitmap = wxBitmap() (see module wxBitmap)
BmpDisabled = wxBitmap() (see module wxBitmap)

See **external documentation**.
Also:
addTool(This, Toolid, Label, Bitmap, [Option]) -> wx:wx_object() when
This::wxToolBar(), Toolid::integer(), Label::unicode:chardata(), Bitmap::wxBitmap:wxBitmap(),
Option :: {shortHelp, unicode:chardata()}
| {kind, wx:wx_enum()};
(This, Toolid, Bitmap, BmpDisabled, [Option]) -> wx:wx_object() when
This::wxToolBar(), Toolid::integer(), Bitmap::wxBitmap:wxBitmap(), BmpDisabled::wxBitmap:wxBitmap(),
Option :: {toggle, boolean()}
| {clientData, wx:wx_object()}
| {shortHelpString, unicode:chardata()}
| {longHelpString, unicode:chardata()}.

Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

addTool(This, Toolid, Bitmap, BmpDisabled, Toggle, XPos) -> wx_object() (see module wx)

Types:

This = wxToolBar()
Toolid = integer()
Bitmap = wxBitmap() (see module wxBitmap)
BmpDisabled = wxBitmap() (see module wxBitmap)
Toggle = boolean()

XPos = integer()

See **external documentation**.
Also:
addTool(This, Toolid, Label, Bitmap, BmpDisabled, [Option]) -> wx:wx_object() when
This::wxToolBar(), Toolid::integer(), Label::unicode:chardata(), Bitmap::wxBitmap:wxBitmap(),
BmpDisabled::wxBitmap:wxBitmap(),
Option :: {kind, wx:wx_enum()}
| {shortHelp, unicode:chardata()}
| {longHelp, unicode:chardata()}
| {data, wx:wx_object()}.

Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

addTool(This, Toolid, Bitmap, BmpDisabled, Toggle, XPos, Option::[Option]) -> wx_object() (see module wx)

Types:

This = wxToolBar()
Toolid = integer()
Bitmap = wxBitmap() (see module wxBitmap)
BmpDisabled = wxBitmap() (see module wxBitmap)
Toggle = boolean()
XPos = integer()
Option = {yPos, integer()} | {clientData, wx_object() (see module wx)} | {shortHelp, chardata() (see module unicode)} | {longHelp, chardata() (see module unicode)}

See **external documentation**.

addCheckTool(This, Toolid, Label, Bitmap) -> wx_object() (see module wx)

Types:

This = wxToolBar()
Toolid = integer()
Label = chardata() (see module unicode)
Bitmap = wxBitmap() (see module wxBitmap)

Equivalent to *addCheckTool(This, Toolid, Label, Bitmap, [])*.

addCheckTool(This, Toolid, Label, Bitmap, Option::[Option]) -> wx_object() (see module wx)

Types:

This = wxToolBar()
Toolid = integer()
Label = chardata() (see module unicode)
Bitmap = wxBitmap() (see module wxBitmap)
Option = {bmpDisabled, wxBitmap() (see module wxBitmap)} | {shortHelp, chardata() (see module unicode)} | {longHelp, chardata() (see module unicode)} | {data, wx_object() (see module wx)}

See **external documentation**.

addRadioTool(This, Toolid, Label, Bitmap) -> wx_object() (see module wx)

Types:

This = wxToolBar()

Toolid = integer()

Label = chardata() (see module unicode)

Bitmap = wxBitmap() (see module wxBitmap)

Equivalent to *addRadioTool(This, Toolid, Label, Bitmap, [])*.

addRadioTool(This, Toolid, Label, Bitmap, Option::[Option]) -> wx_object() (see module wx)

Types:

This = wxToolBar()

Toolid = integer()

Label = chardata() (see module unicode)

Bitmap = wxBitmap() (see module wxBitmap)

Option = {bmpDisabled, wxBitmap() (see module wxBitmap)} | {shortHelp, chardata() (see module unicode)} | {longHelp, chardata() (see module unicode)} | {data, wx_object() (see module wx)}

See **external documentation**.

deleteTool(This, Toolid) -> boolean()

Types:

This = wxToolBar()

Toolid = integer()

See **external documentation**.

deleteToolByPos(This, Pos) -> boolean()

Types:

This = wxToolBar()

Pos = integer()

See **external documentation**.

enableTool(This, Toolid, Enable) -> ok

Types:

This = wxToolBar()

Toolid = integer()

Enable = boolean()

See **external documentation**.

findById(This, Toolid) -> wx_object() (see module wx)

Types:

This = wxToolBar()

```
Toolid = integer()
```

See **external documentation**.

```
findControl(This, Toolid) -> wxControl() (see module wxControl)
```

Types:

```
This = wxToolBar()
Toolid = integer()
```

See **external documentation**.

```
findToolForPosition(This, X, Y) -> wx_object() (see module wx)
```

Types:

```
This = wxToolBar()
X = integer()
Y = integer()
```

See **external documentation**.

```
getToolSize(This) -> {W::integer(), H::integer()}
```

Types:

```
This = wxToolBar()
```

See **external documentation**.

```
getToolBitmapSize(This) -> {W::integer(), H::integer()}
```

Types:

```
This = wxToolBar()
```

See **external documentation**.

```
getMargins(This) -> {W::integer(), H::integer()}
```

Types:

```
This = wxToolBar()
```

See **external documentation**.

```
getToolEnabled(This, Toolid) -> boolean()
```

Types:

```
This = wxToolBar()
Toolid = integer()
```

See **external documentation**.

```
getToolLongHelp(This, Toolid) -> charlist() (see module unicode)
```

Types:

```
This = wxToolBar()
Toolid = integer()
```

See **external documentation**.

getToolPacking(This) -> integer()

Types:

This = wxToolBar()

See **external documentation**.

getToolPos(This, Id) -> integer()

Types:

This = wxToolBar()

Id = integer()

See **external documentation**.

getToolSeparation(This) -> integer()

Types:

This = wxToolBar()

See **external documentation**.

getToolShortHelp(This, Toolid) -> charlist() (see module unicode)

Types:

This = wxToolBar()

Toolid = integer()

See **external documentation**.

getToolState(This, Toolid) -> boolean()

Types:

This = wxToolBar()

Toolid = integer()

See **external documentation**.

insertControl(This, Pos, Control) -> wx_object() (see module wx)

Types:

This = wxToolBar()

Pos = integer()

Control = wxControl() (see module wxControl)

See **external documentation**.

insertSeparator(This, Pos) -> wx_object() (see module wx)

Types:

This = wxToolBar()

Pos = integer()

See **external documentation**.

insertTool(This, Pos, Tool) -> wx_object() (see module wx)

Types:

```
This = wxToolBar()
Pos = integer()
Tool = wx_object() (see module wx)
```

See **external documentation**.

insertTool(This, Pos, Toolid, Bitmap) -> wx_object() (see module wx)

Types:

```
This = wxToolBar()
Pos = integer()
Toolid = integer()
Bitmap = wxBitmap() (see module wxBitmap)
```

Equivalent to *insertTool(This, Pos, Toolid, Bitmap, [])*.

insertTool(This, Pos, Toolid, Label, Bitmap) -> wx_object() (see module wx)

Types:

```
This = wxToolBar()
Pos = integer()
Toolid = integer()
Label = chardata() (see module unicode)
Bitmap = wxBitmap() (see module wxBitmap)
```

See **external documentation**.

Also:

insertTool(This, Pos, Toolid, Bitmap, [Option]) -> wx:wx_object() when

This::wxToolBar(), Pos::integer(), Toolid::integer(), Bitmap::wxBitmap:wxBitmap(),

Option :: {bmpDisabled, wxBitmap:wxBitmap()}

| {toggle, boolean()}

| {clientData, wx:wx_object()}

| {shortHelp, unicode:chardata()}

| {longHelp, unicode:chardata()}.

Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

insertTool(This, Pos, Toolid, Label, Bitmap, Option::[Option]) -> wx_object() (see module wx)

Types:

```
This = wxToolBar()
Pos = integer()
Toolid = integer()
Label = chardata() (see module unicode)
Bitmap = wxBitmap() (see module wxBitmap)
Option = {bmpDisabled, wxBitmap() (see module wxBitmap)} | {kind,
wx_enum() (see module wx)} | {shortHelp, chardata() (see module unicode)}
| {longHelp, chardata() (see module unicode)} | {clientData, wx_object()
(see module wx)}
```

See **external documentation**.

Kind = ?wxITEM_SEPARATOR | ?wxITEM_NORMAL | ?wxITEM_CHECK | ?wxITEM_RADIO | ?wxITEM_MAX

realize(This) -> boolean()

Types:

This = wxToolBar()

See **external documentation**.

removeTool(This, Toolid) -> wx_object() (see module wx)

Types:

This = wxToolBar()
Toolid = integer()

See **external documentation**.

setMargins(This, X, Y) -> ok

Types:

This = wxToolBar()
X = integer()
Y = integer()

See **external documentation**.

setToolBitmapSize(This, Size) -> ok

Types:

This = wxToolBar()
Size = {W::integer(), H::integer()}

See **external documentation**.

setToolLongHelp(This, Toolid, HelpString) -> ok

Types:

This = wxToolBar()
Toolid = integer()
HelpString = chardata() (see module unicode)

See **external documentation**.

setToolPacking(This, Packing) -> ok

Types:

This = wxToolBar()
Packing = integer()

See **external documentation**.

setToolShortHelp(This, Id, HelpString) -> ok

Types:

This = wxToolBar()

```
Id = integer()
HelpString = chardata() (see module unicode)
```

See **external documentation**.

setToolSeparation(This, Separation) -> ok

Types:

```
This = wxToolBar()
Separation = integer()
```

See **external documentation**.

toggleTool(This, Toolid, Toggle) -> ok

Types:

```
This = wxToolBar()
Toolid = integer()
Toggle = boolean()
```

See **external documentation**.

wxToolTip

Erlang module

See external documentation: **wxToolTip**.

DATA TYPES

`wxToolTip()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`enable(Flag) -> ok`

Types:

`Flag = boolean()`

See **external documentation**.

`setDelay(Msecs) -> ok`

Types:

`Msecs = integer()`

See **external documentation**.

`new(Tip) -> wxToolTip()`

Types:

`Tip = chardata() (see module unicode)`

See **external documentation**.

`setTip(This, Tip) -> ok`

Types:

`This = wxToolTip()`

`Tip = chardata() (see module unicode)`

See **external documentation**.

`getTip(This) -> charlist() (see module unicode)`

Types:

`This = wxToolTip()`

See **external documentation**.

`getWindow(This) -> wxWindow() (see module wxWindow)`

Types:

`This = wxToolTip()`

See **external documentation**.

destroy(This::wxToolTip()) -> ok

Destroys this object, do not use object again

wxToolbook

Erlang module

See external documentation: **wxToolbook**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxToolbook()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxToolbook()`

See **external documentation**.

`new(Parent, Id) -> wxToolbook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxToolbook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`addPage(This, Page, Text) -> boolean()`

Types:

```
This = wxToolbook()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
```

Equivalent to *addPage(This, Page, Text, [])*.

`addPage(This, Page, Text, Option::[Option]) -> boolean()`

Types:

```
This = wxToolbook()
```

```
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}
```

See **external documentation**.

advanceSelection(This) -> ok

Types:

```
This = wxToolbook()
```

Equivalent to *advanceSelection(This, [])*.

advanceSelection(This, Option::[Option]) -> ok

Types:

```
This = wxToolbook()
Option = {forward, boolean()}
```

See **external documentation**.

assignImageList(This, ImageList) -> ok

Types:

```
This = wxToolbook()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

create(This, Parent, Id) -> boolean()

Types:

```
This = wxToolbook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

create(This, Parent, Id, Option::[Option]) -> boolean()

Types:

```
This = wxToolbook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

deleteAllPages(This) -> boolean()

Types:

```
This = wxToolbook()
```

See **external documentation**.

deletePage(This, N) -> boolean()

Types:

This = wxToolbook()

N = integer()

See **external documentation**.

removePage(This, N) -> boolean()

Types:

This = wxToolbook()

N = integer()

See **external documentation**.

getCurrentPage(This) -> wxWindow() (see module wxWindow)

Types:

This = wxToolbook()

See **external documentation**.

getImageList(This) -> wxImageList() (see module wxImageList)

Types:

This = wxToolbook()

See **external documentation**.

getPage(This, N) -> wxWindow() (see module wxWindow)

Types:

This = wxToolbook()

N = integer()

See **external documentation**.

getPageCount(This) -> integer()

Types:

This = wxToolbook()

See **external documentation**.

getPageImage(This, N) -> integer()

Types:

This = wxToolbook()

N = integer()

See **external documentation**.

getPageText(This, N) -> charlist() (see module unicode)

Types:

This = wxToolbook()

N = integer()

See **external documentation**.

getSelection(This) -> integer()

Types:

This = wxToolbook()

See **external documentation**.

hitTest(This, Pt) -> Result

Types:

Result = {Res::integer(), Flags::integer()}
This = wxToolbook()
Pt = {X::integer(), Y::integer()}

See **external documentation**.

insertPage(This, N, Page, Text) -> boolean()

Types:

This = wxToolbook()
N = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)

Equivalent to *insertPage(This, N, Page, Text, [])*.

insertPage(This, N, Page, Text, Option::[Option]) -> boolean()

Types:

This = wxToolbook()
N = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}

See **external documentation**.

setImageList(This, ImageList) -> ok

Types:

This = wxToolbook()
ImageList = wxImageList() (see module wxImageList)

See **external documentation**.

setPageSize(This, Size) -> ok

Types:

This = wxToolbook()
Size = {W::integer(), H::integer()}

See **external documentation**.

setPageImage(This, N, ImageId) -> boolean()

Types:

This = wxToolbook()
N = integer()
ImageId = integer()

See **external documentation**.

setPageText(This, N, StrText) -> boolean()

Types:

This = wxToolbook()
N = integer()
StrText = chardata() (see module unicode)

See **external documentation**.

setSelection(This, N) -> integer()

Types:

This = wxToolbook()
N = integer()

See **external documentation**.

changeSelection(This, N) -> integer()

Types:

This = wxToolbook()
N = integer()

See **external documentation**.

destroy(This::wxToolbook()) -> ok

Destroys this object, do not use object again

wxTopLevelWindow

Erlang module

See external documentation: **wxTopLevelWindow**.

This class is derived (and can use functions) from:
wxWindow
wxEvtHandler

DATA TYPES

`wxTopLevelWindow()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getIcon(This) -> wxIcon() (see module wxIcon)`

Types:

`This = wxTopLevelWindow()`

See **external documentation**.

`getIcons(This) -> wxIconBundle() (see module wxIconBundle)`

Types:

`This = wxTopLevelWindow()`

See **external documentation**.

`getTitle(This) -> charlist() (see module unicode)`

Types:

`This = wxTopLevelWindow()`

See **external documentation**.

`isActive(This) -> boolean()`

Types:

`This = wxTopLevelWindow()`

See **external documentation**.

`iconize(This) -> ok`

Types:

`This = wxTopLevelWindow()`

Equivalent to *iconize(This, [])*.

`iconize(This, Option::[Option]) -> ok`

Types:

```
This = wxTopLevelWindow()
Option = {iconize, boolean()}
```

See **external documentation**.

isFullScreen(This) -> boolean()

Types:

```
This = wxTopLevelWindow()
```

See **external documentation**.

isIconized(This) -> boolean()

Types:

```
This = wxTopLevelWindow()
```

See **external documentation**.

isMaximized(This) -> boolean()

Types:

```
This = wxTopLevelWindow()
```

See **external documentation**.

maximize(This) -> ok

Types:

```
This = wxTopLevelWindow()
```

Equivalent to *maximize(This, [])*.

maximize(This, Option::[Option]) -> ok

Types:

```
This = wxTopLevelWindow()
Option = {maximize, boolean()}
```

See **external documentation**.

requestUserAttention(This) -> ok

Types:

```
This = wxTopLevelWindow()
```

Equivalent to *requestUserAttention(This, [])*.

requestUserAttention(This, Option::[Option]) -> ok

Types:

```
This = wxTopLevelWindow()
Option = {flags, integer()}
```

See **external documentation**.

setIcon(This, Icon) -> ok

Types:

```
This = wxTopLevelWindow()
Icon = wxIcon() (see module wxIcon)
```

See **external documentation**.

setIcons(This, Icons) -> ok

Types:

```
This = wxTopLevelWindow()
Icons = wxIconBundle() (see module wxIconBundle)
```

See **external documentation**.

centerOnScreen(This) -> ok

Types:

```
This = wxTopLevelWindow()
```

Equivalent to *centerOnScreen(This, [])*.

centerOnScreen(This, Option::[Option]) -> ok

Types:

```
This = wxTopLevelWindow()
Option = {dir, integer()}
```

See **external documentation**.

centreOnScreen(This) -> ok

Types:

```
This = wxTopLevelWindow()
```

Equivalent to *centreOnScreen(This, [])*.

centreOnScreen(This, Option::[Option]) -> ok

Types:

```
This = wxTopLevelWindow()
Option = {dir, integer()}
```

See **external documentation**.

setShape(This, Region) -> boolean()

Types:

```
This = wxTopLevelWindow()
Region = wxRegion() (see module wxRegion)
```

See **external documentation**.

setTitle(This, Title) -> ok

Types:

```
This = wxTopLevelWindow()
Title = chardata() (see module unicode)
```

See **external documentation**.

showFullScreen(This, Show) -> boolean()

Types:

This = wxTopLevelWindow()

Show = boolean()

Equivalent to *showFullScreen(This, Show, [])*.

showFullScreen(This, Show, Option::[Option]) -> boolean()

Types:

This = wxTopLevelWindow()

Show = boolean()

Option = {style, integer()}

See **external documentation**.

wxTreeCtrl

Erlang module

See external documentation: **wxTreeCtrl**.

Note: The representation of treeItemId() have changed from the original class implementation to be an semi-opaque type,Equality between TreeItemId's can be tested and zero means that the TreeItem is invalid.

DATA TYPES

`wxTreeCtrl()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxTreeCtrl()`

See **external documentation**.

`new(Parent) -> wxTreeCtrl()`

Types:

`Parent = wxWindow() (see module wxWindow)`

Equivalent to *new(Parent, [])*.

`new(Parent, Option::[Option]) -> wxTreeCtrl()`

Types:

`Parent = wxWindow() (see module wxWindow)`

`Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} | {size, {W::integer(), H::integer()}} | {style, integer()} | {validator, wx_object() (see module wx)}`

See **external documentation**.

`addRoot(This, Text) -> integer()`

Types:

`This = wxTreeCtrl()`

`Text = chardata() (see module unicode)`

Equivalent to *addRoot(This, Text, [])*.

`addRoot(This, Text, Option::[Option]) -> integer()`

Types:

`This = wxTreeCtrl()`

`Text = chardata() (see module unicode)`

`Option = {image, integer()} | {selectedImage, integer()} | {data, term()}`

See **external documentation**.

appendItem(This, Parent, Text) -> integer()

Types:

This = wxTreeCtrl()
Parent = integer()
Text = chardata() (see module unicode)

Equivalent to *appendItem(This, Parent, Text, [])*.

appendItem(This, Parent, Text, Option::[Option]) -> integer()

Types:

This = wxTreeCtrl()
Parent = integer()
Text = chardata() (see module unicode)
Option = {image, integer()} | {selectedImage, integer()} | {data, term()}

See **external documentation**.

assignImageList(This, ImageList) -> ok

Types:

This = wxTreeCtrl()
ImageList = wxImageList() (see module wxImageList)

See **external documentation**.

assignStateImageList(This, ImageList) -> ok

Types:

This = wxTreeCtrl()
ImageList = wxImageList() (see module wxImageList)

See **external documentation**.

collapse(This, Item) -> ok

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

collapseAndReset(This, Item) -> ok

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

create(This, Parent) -> boolean()

Types:

This = wxTreeCtrl()
Parent = wxWindow() (see module wxWindow)

Equivalent to *create(This, Parent, [])*.

create(This, Parent, Option::[Option]) -> boolean()

Types:

```
This = wxTreeCtrl()
Parent = wxWindow() (see module wxWindow)
Option = {id, integer()} | {pos, {X::integer(), Y::integer()}} |
{size, {W::integer(), H::integer()}} | {style, integer()} | {validator,
wx_object() (see module wx)}
```

See **external documentation**.

delete(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

deleteAllItems(This) -> ok

Types:

```
This = wxTreeCtrl()
```

See **external documentation**.

deleteChildren(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

editLabel(This, Item) -> wxTextCtrl() (see module wxTextCtrl)

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

ensureVisible(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

expand(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
```

```
Item = integer()
```

See **external documentation**.

getBoundingRect(This, Item, Rect) -> boolean()

Types:

```
This = wxTreeCtrl()
Item = integer()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
```

Equivalent to *getBoundingRect(This, Item, Rect, [])*.

getBoundingRect(This, Item, Rect, Option::[Option]) -> boolean()

Types:

```
This = wxTreeCtrl()
Item = integer()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}
Option = {textOnly, boolean()}
```

See **external documentation**.

getChildrenCount(This, Item) -> integer()

Types:

```
This = wxTreeCtrl()
Item = integer()
```

Equivalent to *getChildrenCount(This, Item, [])*.

getChildrenCount(This, Item, Option::[Option]) -> integer()

Types:

```
This = wxTreeCtrl()
Item = integer()
Option = {recursively, boolean()}
```

See **external documentation**.

getCount(This) -> integer()

Types:

```
This = wxTreeCtrl()
```

See **external documentation**.

getEditControl(This) -> wxTextCtrl() (see module wxTextCtrl)

Types:

```
This = wxTreeCtrl()
```

See **external documentation**.

getFirstChild(This, Item) -> Result

Types:

```
Result = {Res::integer(), Cookie::integer()}
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

getNextChild(This, Item, Cookie) -> Result

Types:

```
Result = {Res::integer(), Cookie::integer()}
This = wxTreeCtrl()
Item = integer()
Cookie = integer()
```

See **external documentation**.

getFirstVisibleItem(This) -> integer()

Types:

```
This = wxTreeCtrl()
```

See **external documentation**.

getImageList(This) -> wxImageList() (see module wxImageList)

Types:

```
This = wxTreeCtrl()
```

See **external documentation**.

getIndent(This) -> integer()

Types:

```
This = wxTreeCtrl()
```

See **external documentation**.

getItemBackgroundColour(This, Item) -> wx_colour4() (see module wx)

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

getItemData(This, Item) -> term()

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

getItemFont(This, Item) -> wxFont() (see module wxFont)

Types:

```
This = wxTreeCtrl()
```

```
Item = integer()
```

See **external documentation**.

getItemImage(This, Item) -> integer()

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

getItemImage(This, Item, Option::[Option]) -> integer()

Types:

```
This = wxTreeCtrl()
Item = integer()
Option = {which, wx_enum() (see module wx)}
```

See **external documentation**.
Which = ?wxTreeItemIcon_Normal | ?wxTreeItemIcon_Selected | ?wxTreeItemIcon_Expanded | ?wxTreeItemIcon_SelectedExpanded | ?wxTreeItemIcon_Max

getItemText(This, Item) -> charlist() (see module unicode)

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

getItemTextColour(This, Item) -> wx_colour4() (see module wx)

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

getLastChild(This, Item) -> integer()

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

getNextSibling(This, Item) -> integer()

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

getNextVisible(This, Item) -> integer()

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

getItemParent(This, Item) -> integer()

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

getPrevSibling(This, Item) -> integer()

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

getPrevVisible(This, Item) -> integer()

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

getRootItem(This) -> integer()

Types:

This = wxTreeCtrl()

See **external documentation**.

getSelection(This) -> integer()

Types:

This = wxTreeCtrl()

See **external documentation**.

getSelections(This) -> Result

Types:

Result = {Res::integer(), Val::[integer()]}
This = wxTreeCtrl()

See **external documentation**.

getStateImageList(This) -> wxImageList() (see module wxImageList)

Types:

This = wxTreeCtrl()

See **external documentation**.

hitTest(This, Point) -> integer()

Types:

```
This = wxTreeCtrl()
Point = {X::integer(), Y::integer()}
```

See **external documentation**.

insertItem(This, Parent, Pos, Text) -> integer()

Types:

```
This = wxTreeCtrl()
Parent = integer()
Pos = integer()
Text = chardata() (see module unicode)
```

Equivalent to *insertItem(This, Parent, Pos, Text, [])*.

insertItem(This, Parent, Pos, Text, Option::[Option]) -> integer()

Types:

```
This = wxTreeCtrl()
Parent = integer()
Pos = integer()
Text = chardata() (see module unicode)
Option = {image, integer()} | {selImage, integer()} | {data, term()}
```

See **external documentation**.

isBold(This, Item) -> boolean()

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

isExpanded(This, Item) -> boolean()

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

isSelected(This, Item) -> boolean()

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

isVisible(This, Item) -> boolean()

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

itemHasChildren(This, Item) -> boolean()

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

prependItem(This, Parent, Text) -> integer()

Types:

This = wxTreeCtrl()
Parent = integer()
Text = chardata() (see module unicode)

Equivalent to *prependItem(This, Parent, Text, [])*.

prependItem(This, Parent, Text, Option::[Option]) -> integer()

Types:

This = wxTreeCtrl()
Parent = integer()
Text = chardata() (see module unicode)
Option = {image, integer()} | {selectedImage, integer()} | {data, term()}

See **external documentation**.

scrollTo(This, Item) -> ok

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

selectItem(This, Item) -> ok

Types:

This = wxTreeCtrl()
Item = integer()

See **external documentation**.

selectItem(This, Item, Option::[Option]) -> ok

Types:

This = wxTreeCtrl()
Item = integer()

```
Option = {select, boolean()}
```

See **external documentation**.

setIndent(This, Indent) -> ok

Types:

```
This = wxTreeCtrl()
Indent = integer()
```

See **external documentation**.

setImageList(This, ImageList) -> ok

Types:

```
This = wxTreeCtrl()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

setItemBackgroundColour(This, Item, Col) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Col = wx_colour() (see module wx)
```

See **external documentation**.

setItemBold(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

Equivalent to *setItemBold(This, Item, [])*.

setItemBold(This, Item, Option::[Option]) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Option = {bold, boolean()}
```

See **external documentation**.

setItemData(This, Item, Data) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Data = term()
```

See **external documentation**.

setItemDropHighlight(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

Equivalent to *setItemDropHighlight(This, Item, [])*.

setItemDropHighlight(This, Item, Option::[Option]) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Option = {highlight, boolean()}
```

See **external documentation**.

setItemFont(This, Item, Font) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Font = wxFont() (see module wxFont)
```

See **external documentation**.

setItemHasChildren(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

Equivalent to *setItemHasChildren(This, Item, [])*.

setItemHasChildren(This, Item, Option::[Option]) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Option = {has, boolean()}
```

See **external documentation**.

setItemImage(This, Item, Image) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Image = integer()
```

See **external documentation**.

setItemImage(This, Item, Image, Option::[Option]) -> ok

Types:

```
This = wxTreeCtrl()
```

```
Item = integer()
Image = integer()
Option = {which, wx_enum() (see module wx)}
```

See **external documentation**.
Which = ?wxTreeItemIcon_Normal | ?wxTreeItemIcon_Selected | ?wxTreeItemIcon_Expanded | ?wxTreeItemIcon_SelectedExpanded | ?wxTreeItemIcon_Max

setItemText(This, Item, Text) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Text = chardata() (see module unicode)
```

See **external documentation**.

setItemTextColour(This, Item, Col) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
Col = wx_colour() (see module wx)
```

See **external documentation**.

setStateImageList(This, ImageList) -> ok

Types:

```
This = wxTreeCtrl()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

setWindowStyle(This, Styles) -> ok

Types:

```
This = wxTreeCtrl()
Styles = integer()
```

See **external documentation**.

sortChildren(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

toggle(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

toggleItemSelection(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

unselect(This) -> ok

Types:

```
This = wxTreeCtrl()
```

See **external documentation**.

unselectAll(This) -> ok

Types:

```
This = wxTreeCtrl()
```

See **external documentation**.

unselectItem(This, Item) -> ok

Types:

```
This = wxTreeCtrl()
Item = integer()
```

See **external documentation**.

destroy(This::wxTreeCtrl()) -> ok

Destroys this object, do not use object again

wxTreeEvent

Erlang module

See external documentation: **wxTreeEvent**.

Use *wxEvtHandler:connect/3* with EventType:

command_tree_begin_drag, *command_tree_begin_rdrag*, *command_tree_begin_label_edit*, *command_tree_end_label_edit*, *command_tree_delete_item*, *command_tree_get_info*, *command_tree_set_info*, *command_tree_item_expanded*, *command_tree_item_expanding*, *command_tree_item_collapsed*, *command_tree_item_collapsing*, *command_tree_sel_changed*, *command_tree_sel_changing*, *command_tree_key_down*, *command_tree_item_activated*, *command_tree_item_right_click*, *command_tree_item_middle_click*, *command_tree_end_drag*, *command_tree_state_image_click*, *command_tree_item_gettooltip*, *command_tree_item_menu*

See also the message variant *#wxTree{}* event record type.

This class is derived (and can use functions) from:
wxNotifyEvent
wxCommandEvent
wxEvent

DATA TYPES

`wxTreeEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`getKeyCode(This) -> integer()`

Types:

`This = wxTreeEvent()`

See **external documentation**.

`getItem(This) -> integer()`

Types:

`This = wxTreeEvent()`

See **external documentation**.

`getKeyEvent(This) -> wxKeyEvent() (see module wxKeyEvent)`

Types:

`This = wxTreeEvent()`

See **external documentation**.

`getLabel(This) -> charlist() (see module unicode)`

Types:

`This = wxTreeEvent()`

See **external documentation**.

getOldItem(This) -> integer()

Types:

This = wxTreeEvent()

See **external documentation**.

getPoint(This) -> {X::integer(), Y::integer()}

Types:

This = wxTreeEvent()

See **external documentation**.

isEditCancelled(This) -> boolean()

Types:

This = wxTreeEvent()

See **external documentation**.

setToolTip(This, ToolTip) -> ok

Types:

This = wxTreeEvent()

ToolTip = chardata() (see module unicode)

See **external documentation**.

wxTreebook

Erlang module

See external documentation: **wxTreebook**.

This class is derived (and can use functions) from:
wxControl
wxWindow
wxEvtHandler

DATA TYPES

`wxTreebook()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxTreebook()`

See **external documentation**.

`new(Parent, Id) -> wxTreebook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxTreebook()`

Types:

```
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`addPage(This, Page, Text) -> boolean()`

Types:

```
This = wxTreebook()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
```

Equivalent to *addPage(This, Page, Text, [])*.

`addPage(This, Page, Text, Option::[Option]) -> boolean()`

Types:

```
This = wxTreebook()
```

```
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}
```

See **external documentation**.

advanceSelection(This) -> ok

Types:

```
This = wxTreebook()
```

Equivalent to *advanceSelection(This, [])*.

advanceSelection(This, Option::[Option]) -> ok

Types:

```
This = wxTreebook()
Option = {forward, boolean()}
```

See **external documentation**.

assignImageList(This, ImageList) -> ok

Types:

```
This = wxTreebook()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

create(This, Parent, Id) -> boolean()

Types:

```
This = wxTreebook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
```

Equivalent to *create(This, Parent, Id, [])*.

create(This, Parent, Id, Option::[Option]) -> boolean()

Types:

```
This = wxTreebook()
Parent = wxWindow() (see module wxWindow)
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

deleteAllPages(This) -> boolean()

Types:

```
This = wxTreebook()
```

See **external documentation**.

deletePage(This, Pos) -> boolean()

Types:

This = wxTreebook()

Pos = integer()

See **external documentation**.

removePage(This, N) -> boolean()

Types:

This = wxTreebook()

N = integer()

See **external documentation**.

getCurrentPage(This) -> wxWindow() (see module wxWindow)

Types:

This = wxTreebook()

See **external documentation**.

getImageList(This) -> wxImageList() (see module wxImageList)

Types:

This = wxTreebook()

See **external documentation**.

getPage(This, N) -> wxWindow() (see module wxWindow)

Types:

This = wxTreebook()

N = integer()

See **external documentation**.

getPageCount(This) -> integer()

Types:

This = wxTreebook()

See **external documentation**.

getPageImage(This, N) -> integer()

Types:

This = wxTreebook()

N = integer()

See **external documentation**.

getPageText(This, N) -> charlist() (see module unicode)

Types:

This = wxTreebook()

N = integer()

See **external documentation**.

getSelection(This) -> integer()

Types:

This = wxTreebook()

See **external documentation**.

expandNode(This, Pos) -> boolean()

Types:

This = wxTreebook()

Pos = integer()

Equivalent to *expandNode(This, Pos, [])*.

expandNode(This, Pos, Option::[Option]) -> boolean()

Types:

This = wxTreebook()

Pos = integer()

Option = {expand, boolean()}

See **external documentation**.

isNodeExpanded(This, Pos) -> boolean()

Types:

This = wxTreebook()

Pos = integer()

See **external documentation**.

hitTest(This, Pt) -> Result

Types:

Result = {Res::integer(), Flags::integer()}

This = wxTreebook()

Pt = {X::integer(), Y::integer()}

See **external documentation**.

insertPage(This, Pos, Page, Text) -> boolean()

Types:

This = wxTreebook()

Pos = integer()

Page = wxWindow() (see module wxWindow)

Text = chardata() (see module unicode)

Equivalent to *insertPage(This, Pos, Page, Text, [])*.

insertPage(This, Pos, Page, Text, Option::[Option]) -> boolean()

Types:

```
This = wxTreebook()
Pos = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}
```

See **external documentation**.

insertSubPage(This, Pos, Page, Text) -> boolean()

Types:

```
This = wxTreebook()
Pos = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
```

Equivalent to *insertSubPage(This, Pos, Page, Text, [])*.

insertSubPage(This, Pos, Page, Text, Option::[Option]) -> boolean()

Types:

```
This = wxTreebook()
Pos = integer()
Page = wxWindow() (see module wxWindow)
Text = chardata() (see module unicode)
Option = {bSelect, boolean()} | {imageId, integer()}
```

See **external documentation**.

setImageList(This, ImageList) -> ok

Types:

```
This = wxTreebook()
ImageList = wxImageList() (see module wxImageList)
```

See **external documentation**.

setPageSize(This, Size) -> ok

Types:

```
This = wxTreebook()
Size = {W::integer(), H::integer()}
```

See **external documentation**.

setPageImage(This, N, ImageId) -> boolean()

Types:

```
This = wxTreebook()
N = integer()
ImageId = integer()
```

See **external documentation**.

setPageText(This, N, StrText) -> boolean()

Types:

This = wxTreebook()
N = integer()
StrText = chardata() (see module unicode)

See **external documentation**.

setSelection(This, N) -> integer()

Types:

This = wxTreebook()
N = integer()

See **external documentation**.

changeSelection(This, N) -> integer()

Types:

This = wxTreebook()
N = integer()

See **external documentation**.

destroy(This::wxTreebook()) -> ok

Destroys this object, do not use object again

wxUpdateUIEvent

Erlang module

See external documentation: **wxUpdateUIEvent**.

Use *wxEvtHandler:connect/3* with EventType:

update_ui

See also the message variant *#wxUpdateUI{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxUpdateUIEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`canUpdate(Win) -> boolean()`

Types:

`Win = wxWindow() (see module wxWindow)`

See **external documentation**.

`check(This, Check) -> ok`

Types:

`This = wxUpdateUIEvent()`
`Check = boolean()`

See **external documentation**.

`enable(This, Enable) -> ok`

Types:

`This = wxUpdateUIEvent()`
`Enable = boolean()`

See **external documentation**.

`show(This, Show) -> ok`

Types:

`This = wxUpdateUIEvent()`
`Show = boolean()`

See **external documentation**.

getChecked(This) -> boolean()

Types:

This = wxUpdateUIEvent()

See **external documentation**.

getEnabled(This) -> boolean()

Types:

This = wxUpdateUIEvent()

See **external documentation**.

getShown(This) -> boolean()

Types:

This = wxUpdateUIEvent()

See **external documentation**.

getSetChecked(This) -> boolean()

Types:

This = wxUpdateUIEvent()

See **external documentation**.

getSetEnabled(This) -> boolean()

Types:

This = wxUpdateUIEvent()

See **external documentation**.

getSetShown(This) -> boolean()

Types:

This = wxUpdateUIEvent()

See **external documentation**.

getSetText(This) -> boolean()

Types:

This = wxUpdateUIEvent()

See **external documentation**.

getText(This) -> charlist() (see module unicode)

Types:

This = wxUpdateUIEvent()

See **external documentation**.

getMode() -> wx_enum() (see module wx)

See **external documentation**.

Res = ?wxUPDATE_UI_PROCESS_ALL | ?wxUPDATE_UI_PROCESS_SPECIFIED

`getUpdateInterval() -> integer()`

See **external documentation**.

`resetUpdateTime() -> ok`

See **external documentation**.

`setMode(Mode) -> ok`

Types:

`Mode = wx_enum() (see module wx)`

See **external documentation**.
Mode = ?wxUPDATE_UI_PROCESS_ALL | ?wxUPDATE_UI_PROCESS_SPECIFIED

`setText(This, Text) -> ok`

Types:

`This = wxUpdateUIEvent()`

`Text = chardata() (see module unicode)`

See **external documentation**.

`setUpdateInterval(UpdateInterval) -> ok`

Types:

`UpdateInterval = integer()`

See **external documentation**.

wxWindow

Erlang module

See external documentation: **wxWindow**.

This class is derived (and can use functions) from:
wxEvtHandler

DATA TYPES

`wxWindow()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxWindow()`

See **external documentation**.

`new(Parent, Id) -> wxWindow()`

Types:

```
Parent = wxWindow()
Id = integer()
```

Equivalent to *new(Parent, Id, [])*.

`new(Parent, Id, Option::[Option]) -> wxWindow()`

Types:

```
Parent = wxWindow()
Id = integer()
Option = {pos, {X::integer(), Y::integer()}} | {size, {W::integer(),
H::integer()}} | {style, integer()}
```

See **external documentation**.

`cacheBestSize(This, Size) -> ok`

Types:

```
This = wxWindow()
Size = {W::integer(), H::integer()}
```

See **external documentation**.

`captureMouse(This) -> ok`

Types:

```
This = wxWindow()
```

See **external documentation**.

center(This) -> ok

Types:

This = wxWindow()

Equivalent to *center(This, [])*.

center(This, Option::[Option]) -> ok

Types:

This = wxWindow()

Option = {dir, integer()}

See **external documentation**.

centerOnParent(This) -> ok

Types:

This = wxWindow()

Equivalent to *centerOnParent(This, [])*.

centerOnParent(This, Option::[Option]) -> ok

Types:

This = wxWindow()

Option = {dir, integer()}

See **external documentation**.

centre(This) -> ok

Types:

This = wxWindow()

Equivalent to *centre(This, [])*.

centre(This, Option::[Option]) -> ok

Types:

This = wxWindow()

Option = {dir, integer()}

See **external documentation**.

centreOnParent(This) -> ok

Types:

This = wxWindow()

Equivalent to *centreOnParent(This, [])*.

centreOnParent(This, Option::[Option]) -> ok

Types:

This = wxWindow()

Option = {dir, integer()}

See **external documentation**.

```
clearBackground(This) -> ok
```

Types:

```
This = wxWindow()
```

See **external documentation**.

```
clientToScreen(This, Pt) -> {X::integer(), Y::integer()}
```

Types:

```
This = wxWindow()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

```
clientToScreen(This, X, Y) -> {X::integer(), Y::integer()}
```

Types:

```
This = wxWindow()
X = integer()
Y = integer()
```

See **external documentation**.

```
close(This) -> boolean()
```

Types:

```
This = wxWindow()
```

Equivalent to *close(This, [])*.

```
close(This, Option::[Option]) -> boolean()
```

Types:

```
This = wxWindow()
Option = {force, boolean()}
```

See **external documentation**.

```
convertDialogToPixels(This, Sz) -> {W::integer(), H::integer()}
```

Types:

```
This = wxWindow()
Sz = {W::integer(), H::integer()}
```

See **external documentation**.

```
convertPixelsToDialog(This, Sz) -> {W::integer(), H::integer()}
```

Types:

```
This = wxWindow()
Sz = {W::integer(), H::integer()}
```

See **external documentation**.

```
Destroy(This) -> boolean()
```

Types:

This = wxWindow()

See **external documentation**.

destroyChildren(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

disable(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

enable(This) -> boolean()

Types:

This = wxWindow()

Equivalent to *enable(This, [])*.

enable(This, Option::[Option]) -> boolean()

Types:

This = wxWindow()

Option = {enable, boolean()}

See **external documentation**.

findFocus() -> wxWindow()

See **external documentation**.

findWindow(This, Winid) -> wxWindow()

Types:

This = wxWindow()

Winid = integer()

See **external documentation**.
Also:
findWindow(This, Name) -> wxWindow() when
This::wxWindow(), Name::unicode:chardata().

findWindowById(Winid) -> wxWindow()

Types:

Winid = integer()

Equivalent to *findWindowById(Winid, [])*.

findWindowById(Winid, Option::[Option]) -> wxWindow()

Types:

```
Winid = integer()
Option = {parent, wxWindow()}
```

See **external documentation**.

findWindowByName(Name) -> wxWindow()

Types:

```
Name = chardata() (see module unicode)
```

Equivalent to *findWindowByName(Name, [])*.

findWindowByName(Name, Option::[Option]) -> wxWindow()

Types:

```
Name = chardata() (see module unicode)
Option = {parent, wxWindow()}
```

See **external documentation**.

findWindowByLabel(Label) -> wxWindow()

Types:

```
Label = chardata() (see module unicode)
```

Equivalent to *findWindowByLabel(Label, [])*.

findWindowByLabel(Label, Option::[Option]) -> wxWindow()

Types:

```
Label = chardata() (see module unicode)
Option = {parent, wxWindow()}
```

See **external documentation**.

fit(This) -> ok

Types:

```
This = wxWindow()
```

See **external documentation**.

fitInside(This) -> ok

Types:

```
This = wxWindow()
```

See **external documentation**.

freeze(This) -> ok

Types:

```
This = wxWindow()
```

See **external documentation**.

getAcceleratorTable(This) -> wxAcceleratorTable() (see module wxAcceleratorTable)

Types:

This = wxWindow()

See **external documentation**.

getBackgroundColour(This) -> wx_colour4() (see module wx)

Types:

This = wxWindow()

See **external documentation**.

getBackgroundStyle(This) -> wx_enum() (see module wx)

Types:

This = wxWindow()

See **external documentation**.
Res = ?wxBG_STYLE_SYSTEM | ?wxBG_STYLE_COLOUR | ?wxBG_STYLE_CUSTOM

getBestSize(This) -> {W::integer(), H::integer()}

Types:

This = wxWindow()

See **external documentation**.

getCaret(This) -> wxCaret() (see module wxCaret)

Types:

This = wxWindow()

See **external documentation**.

getCapture() -> wxWindow()

See **external documentation**.

getCharHeight(This) -> integer()

Types:

This = wxWindow()

See **external documentation**.

getCharWidth(This) -> integer()

Types:

This = wxWindow()

See **external documentation**.

getChildren(This) -> [wxWindow()]

Types:

This = wxWindow()

See **external documentation**.

getClientSize(This) -> {W::integer(), H::integer()}

Types:

This = wxWindow()

See **external documentation**.

getContainingSizer(This) -> wxSizer() (see module wxSizer)

Types:

This = wxWindow()

See **external documentation**.

getCursor(This) -> wxCursor() (see module wxCursor)

Types:

This = wxWindow()

See **external documentation**.

getDropTarget(This) -> wx_object() (see module wx)

Types:

This = wxWindow()

See **external documentation**.

getEventHandler(This) -> wxEvtHandler() (see module wxEvtHandler)

Types:

This = wxWindow()

See **external documentation**.

getExtraStyle(This) -> integer()

Types:

This = wxWindow()

See **external documentation**.

getFont(This) -> wxFont() (see module wxFont)

Types:

This = wxWindow()

See **external documentation**.

getForegroundColour(This) -> wx_colour4() (see module wx)

Types:

This = wxWindow()

See **external documentation**.

getGrandParent(This) -> wxWindow()

Types:

This = wxWindow()

See **external documentation**.

getHandle(This) -> integer()

Types:

This = wxWindow()

See **external documentation**.

getHelpText(This) -> charlist() (see module unicode)

Types:

This = wxWindow()

See **external documentation**.

getId(This) -> integer()

Types:

This = wxWindow()

See **external documentation**.

getLabel(This) -> charlist() (see module unicode)

Types:

This = wxWindow()

See **external documentation**.

getMaxSize(This) -> {W::integer(), H::integer()}

Types:

This = wxWindow()

See **external documentation**.

getMinSize(This) -> {W::integer(), H::integer()}

Types:

This = wxWindow()

See **external documentation**.

getName(This) -> charlist() (see module unicode)

Types:

This = wxWindow()

See **external documentation**.

getParent(This) -> wxWindow()

Types:

This = wxWindow()

See **external documentation**.

```
getPosition(This) -> {X::integer(), Y::integer()}
```

Types:

```
This = wxWindow()
```

See **external documentation**.

```
getRect(This) -> {X::integer(), Y::integer(), W::integer(), H::integer()}
```

Types:

```
This = wxWindow()
```

See **external documentation**.

```
getScreenPosition(This) -> {X::integer(), Y::integer()}
```

Types:

```
This = wxWindow()
```

See **external documentation**.

```
getScreenRect(This) -> {X::integer(), Y::integer(), W::integer(),
H::integer()}
```

Types:

```
This = wxWindow()
```

See **external documentation**.

```
getScrollPos(This, Orient) -> integer()
```

Types:

```
This = wxWindow()
Orient = integer()
```

See **external documentation**.

```
getScrollRange(This, Orient) -> integer()
```

Types:

```
This = wxWindow()
Orient = integer()
```

See **external documentation**.

```
getScrollThumb(This, Orient) -> integer()
```

Types:

```
This = wxWindow()
Orient = integer()
```

See **external documentation**.

```
getSize(This) -> {W::integer(), H::integer()}
```

Types:

```
This = wxWindow()
```

See **external documentation**.

`getSizer(This) -> wxSizer() (see module wxSizer)`

Types:

```
This = wxWindow()
```

See **external documentation**.

`getTextExtent(This, String) -> Result`

Types:

```
Result = {X::integer(), Y::integer(), Descent::integer(),
ExternalLeading::integer()}
This = wxWindow()
String = chardata() (see module unicode)
```

Equivalent to *getTextExtent(This, String, [])*.

`getTextExtent(This, String, Option::[Option]) -> Result`

Types:

```
Result = {X::integer(), Y::integer(), Descent::integer(),
ExternalLeading::integer()}
This = wxWindow()
String = chardata() (see module unicode)
Option = {theFont, wxFont() (see module wxFont)}
```

See **external documentation**.

`getToolTip(This) -> wxToolTip() (see module wxToolTip)`

Types:

```
This = wxWindow()
```

See **external documentation**.

`getUpdateRegion(This) -> wxRegion() (see module wxRegion)`

Types:

```
This = wxWindow()
```

See **external documentation**.

`getVirtualSize(This) -> {W::integer(), H::integer()}`

Types:

```
This = wxWindow()
```

See **external documentation**.

`getWindowStyleFlag(This) -> integer()`

Types:

```
This = wxWindow()
```

See **external documentation**.

getWindowVariant(This) -> wx_enum() (see module wx)

Types:

This = wxWindow()

See **external documentation**.

Res = ?wxWINDOW_VARIANT_NORMAL | ?wxWINDOW_VARIANT_SMALL | ?wxWINDOW_VARIANT_MINI | ?wxWINDOW_VARIANT_LARGE | ?wxWINDOW_VARIANT_MAX

hasCapture(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

hasScrollbar(This, Orient) -> boolean()

Types:

This = wxWindow()

Orient = integer()

See **external documentation**.

hasTransparentBackground(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

hide(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

inheritAttributes(This) -> ok

Types:

This = wxWindow()

See **external documentation**.

initDialog(This) -> ok

Types:

This = wxWindow()

See **external documentation**.

invalidateBestSize(This) -> ok

Types:

This = wxWindow()

See **external documentation**.

isEnabled(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

isExposed(This, Pt) -> boolean()

Types:

This = wxWindow()

Pt = {X::integer(), Y::integer()}

See **external documentation**.
Also:
isExposed(This, Rect) -> boolean() when
This::wxWindow(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

isExposed(This, X, Y) -> boolean()

Types:

This = wxWindow()

X = integer()

Y = integer()

See **external documentation**.

isExposed(This, X, Y, W, H) -> boolean()

Types:

This = wxWindow()

X = integer()

Y = integer()

W = integer()

H = integer()

See **external documentation**.

isRetained(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

isShown(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

isTopLevel(This) -> boolean()

Types:

```
This = wxWindow()
```

See **external documentation**.

layout(This) -> boolean()

Types:

```
This = wxWindow()
```

See **external documentation**.

lineDown(This) -> boolean()

Types:

```
This = wxWindow()
```

See **external documentation**.

lineUp(This) -> boolean()

Types:

```
This = wxWindow()
```

See **external documentation**.

lower(This) -> ok

Types:

```
This = wxWindow()
```

See **external documentation**.

makeModal(This) -> ok

Types:

```
This = wxWindow()
```

Equivalent to *makeModal(This, [])*.

makeModal(This, Option::[Option]) -> ok

Types:

```
This = wxWindow()
Option = {modal, boolean()}
```

See **external documentation**.

move(This, Pt) -> ok

Types:

```
This = wxWindow()
Pt = {X::integer(), Y::integer()}
```

Equivalent to *move(This, Pt, [])*.

move(This, X, Y) -> ok

Types:

```
This = wxWindow()
```

```
X = integer()
Y = integer()
```

See **external documentation**.
Also:
move(This, Pt, [Option]) -> ok when
This::wxWindow(), Pt::{X::integer(), Y::integer()},
Option :: {flags, integer()}.

move(This, X, Y, Option::[Option]) -> ok

Types:

```
This = wxWindow()
X = integer()
Y = integer()
Option = {flags, integer()}
```

See **external documentation**.

moveAfterInTabOrder(This, Win) -> ok

Types:

```
This = wxWindow()
Win = wxWindow()
```

See **external documentation**.

moveBeforeInTabOrder(This, Win) -> ok

Types:

```
This = wxWindow()
Win = wxWindow()
```

See **external documentation**.

navigate(This) -> boolean()

Types:

```
This = wxWindow()
```

Equivalent to *navigate(This, [])*.

navigate(This, Option::[Option]) -> boolean()

Types:

```
This = wxWindow()
Option = {flags, integer()}
```

See **external documentation**.

pageDown(This) -> boolean()

Types:

```
This = wxWindow()
```

See **external documentation**.

pageUp(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

popEventHandler(This) -> wxEvtHandler() (see module wxEvtHandler)

Types:

This = wxWindow()

Equivalent to *popEventHandler(This, [])*.

popEventHandler(This, Option::[Option]) -> wxEvtHandler() (see module wxEvtHandler)

Types:

This = wxWindow()

Option = {deleteHandler, boolean()}

See **external documentation**.

popupMenu(This, Menu) -> boolean()

Types:

This = wxWindow()

Menu = wxMenu() (see module wxMenu)

Equivalent to *popupMenu(This, Menu, [])*.

popupMenu(This, Menu, Option::[Option]) -> boolean()

Types:

This = wxWindow()

Menu = wxMenu() (see module wxMenu)

Option = {pos, {X::integer(), Y::integer()}}

See **external documentation**.

popupMenu(This, Menu, X, Y) -> boolean()

Types:

This = wxWindow()

Menu = wxMenu() (see module wxMenu)

X = integer()

Y = integer()

See **external documentation**.

raise(This) -> ok

Types:

This = wxWindow()

See **external documentation**.

refresh(This) -> ok

Types:

This = wxWindow()

Equivalent to *refresh(This, [])*.

refresh(This, Option::[Option]) -> ok

Types:

This = wxWindow()

Option = {eraseBackground, boolean()} | {rect, {X::integer(), Y::integer(), W::integer(), H::integer()}}

See **external documentation**.

refreshRect(This, Rect) -> ok

Types:

This = wxWindow()

Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}

Equivalent to *refreshRect(This, Rect, [])*.

refreshRect(This, Rect, Option::[Option]) -> ok

Types:

This = wxWindow()

Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}

Option = {eraseBackground, boolean()}

See **external documentation**.

releaseMouse(This) -> ok

Types:

This = wxWindow()

See **external documentation**.

removeChild(This, Child) -> ok

Types:

This = wxWindow()

Child = wxWindow()

See **external documentation**.

reparent(This, NewParent) -> boolean()

Types:

This = wxWindow()

NewParent = wxWindow()

See **external documentation**.

```
screenToClient(This) -> {X::integer(), Y::integer()}
```

Types:

```
This = wxWindow()
```

See **external documentation**.

```
screenToClient(This, Pt) -> {X::integer(), Y::integer()}
```

Types:

```
This = wxWindow()
Pt = {X::integer(), Y::integer()}
```

See **external documentation**.

```
scrollLines(This, Lines) -> boolean()
```

Types:

```
This = wxWindow()
Lines = integer()
```

See **external documentation**.

```
scrollPages(This, Pages) -> boolean()
```

Types:

```
This = wxWindow()
Pages = integer()
```

See **external documentation**.

```
scrollWindow(This, Dx, Dy) -> ok
```

Types:

```
This = wxWindow()
Dx = integer()
Dy = integer()
```

Equivalent to *scrollWindow(This, Dx, Dy, [])*.

```
scrollWindow(This, Dx, Dy, Option::[Option]) -> ok
```

Types:

```
This = wxWindow()
Dx = integer()
Dy = integer()
Option = {rect, {X::integer(), Y::integer(), W::integer(), H::integer()}}
```

See **external documentation**.

```
setAcceleratorTable(This, Accel) -> ok
```

Types:

```
This = wxWindow()
Accel = wxAcceleratorTable() (see module wxAcceleratorTable)
```

See **external documentation**.

setAutoLayout(This, AutoLayout) -> ok

Types:

This = wxWindow()
AutoLayout = boolean()

See **external documentation**.

setBackgroundColour(This, Colour) -> boolean()

Types:

This = wxWindow()
Colour = wx_colour() (see module wx)

See **external documentation**.

setBackgroundStyle(This, Style) -> boolean()

Types:

This = wxWindow()
Style = wx_enum() (see module wx)

See **external documentation**.
Style = ?wxBG_STYLE_SYSTEM | ?wxBG_STYLE_COLOUR | ?wxBG_STYLE_CUSTOM

setCaret(This, Caret) -> ok

Types:

This = wxWindow()
Caret = wxCaret() (see module wxCaret)

See **external documentation**.

setClientSize(This, Size) -> ok

Types:

This = wxWindow()
Size = {W::integer(), H::integer()}

See **external documentation**.
Also:
setClientSize(This, Rect) -> ok when
This::wxWindow(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()}.

setClientSize(This, Width, Height) -> ok

Types:

This = wxWindow()
Width = integer()
Height = integer()

See **external documentation**.

setContainingSizer(This, Sizer) -> ok

Types:

This = wxWindow()

```
Sizer = wxSizer() (see module wxSizer)
```

See **external documentation**.

setCursor(This, Cursor) -> boolean()

Types:

```
This = wxWindow()
Cursor = wxCursor() (see module wxCursor)
```

See **external documentation**.

setMaxSize(This, MaxSize) -> ok

Types:

```
This = wxWindow()
MaxSize = {W::integer(), H::integer()}
```

See **external documentation**.

setMinSize(This, MinSize) -> ok

Types:

```
This = wxWindow()
MinSize = {W::integer(), H::integer()}
```

See **external documentation**.

setOwnBackgroundColour(This, Colour) -> ok

Types:

```
This = wxWindow()
Colour = wx_colour() (see module wx)
```

See **external documentation**.

setOwnFont(This, Font) -> ok

Types:

```
This = wxWindow()
Font = wxFont() (see module wxFont)
```

See **external documentation**.

setOwnForegroundColour(This, Colour) -> ok

Types:

```
This = wxWindow()
Colour = wx_colour() (see module wx)
```

See **external documentation**.

setDropTarget(This, DropTarget) -> ok

Types:

```
This = wxWindow()
DropTarget = wx_object() (see module wx)
```

See **external documentation**.

setExtraStyle(This, ExStyle) -> ok

Types:

```
This = wxWindow()
ExStyle = integer()
```

See **external documentation**.

setFocus(This) -> ok

Types:

```
This = wxWindow()
```

See **external documentation**.

setFocusFromKbd(This) -> ok

Types:

```
This = wxWindow()
```

See **external documentation**.

setFont(This, Font) -> boolean()

Types:

```
This = wxWindow()
Font = wxFont() (see module wxFont)
```

See **external documentation**.

setForegroundColour(This, Colour) -> boolean()

Types:

```
This = wxWindow()
Colour = wx_colour() (see module wx)
```

See **external documentation**.

setHelpText(This, Text) -> ok

Types:

```
This = wxWindow()
Text = chardata() (see module unicode)
```

See **external documentation**.

setId(This, Winid) -> ok

Types:

```
This = wxWindow()
Winid = integer()
```

See **external documentation**.

setLabel(This, Label) -> ok

Types:

```
This = wxWindow()
Label = chardata() (see module unicode)
```

See **external documentation**.

setName(This, Name) -> ok

Types:

```
This = wxWindow()
Name = chardata() (see module unicode)
```

See **external documentation**.

setPalette(This, Pal) -> ok

Types:

```
This = wxWindow()
Pal = wxPalette() (see module wxPalette)
```

See **external documentation**.

setScrollbar(This, Orient, Pos, ThumbVisible, Range) -> ok

Types:

```
This = wxWindow()
Orient = integer()
Pos = integer()
ThumbVisible = integer()
Range = integer()
```

Equivalent to *setScrollbar(This, Orient, Pos, ThumbVisible, Range, [])*.

setScrollbar(This, Orient, Pos, ThumbVisible, Range, Option::[Option]) -> ok

Types:

```
This = wxWindow()
Orient = integer()
Pos = integer()
ThumbVisible = integer()
Range = integer()
Option = {refresh, boolean()}
```

See **external documentation**.

setScrollPos(This, Orient, Pos) -> ok

Types:

```
This = wxWindow()
Orient = integer()
Pos = integer()
```

Equivalent to *setScrollPos(This, Orient, Pos, [])*.

setScrollPos(This, Orient, Pos, Option::[Option]) -> ok

Types:

This = wxWindow()
Orient = integer()
Pos = integer()
Option = {refresh, boolean()}

See **external documentation**.

setSize(This, Rect) -> ok

Types:

This = wxWindow()
Rect = {X::integer(), Y::integer(), W::integer(), H::integer()}

See **external documentation**.
Also:
setSize(This, Size) -> ok when
This::wxWindow(), Size::{W::integer(), H::integer()}.

setSize(This, Width, Height) -> ok

Types:

This = wxWindow()
Width = integer()
Height = integer()

See **external documentation**.
Also:
setSize(This, Rect, [Option]) -> ok when
This::wxWindow(), Rect::{X::integer(), Y::integer(), W::integer(), H::integer()},
Option :: {sizeFlags, integer()}.

setSize(This, X, Y, Width, Height) -> ok

Types:

This = wxWindow()
X = integer()
Y = integer()
Width = integer()
Height = integer()

Equivalent to *setSize(This, X, Y, Width, Height, [])*.

setSize(This, X, Y, Width, Height, Option::[Option]) -> ok

Types:

This = wxWindow()
X = integer()
Y = integer()
Width = integer()
Height = integer()

```
Option = {sizeFlags, integer()}
```

See **external documentation**.

setSizeHints(This, MinSize) -> ok

Types:

```
This = wxWindow()
MinSize = {W::integer(), H::integer()}
```

Equivalent to *setSizeHints(This, MinSize, [])*.

setSizeHints(This, MinW, MinH) -> ok

Types:

```
This = wxWindow()
MinW = integer()
MinH = integer()
```

See **external documentation**.
Also:
setSizeHints(This, MinSize, [Option]) -> ok when
This::wxWindow(), MinSize::{W::integer(), H::integer()},
Option :: {maxSize, {W::integer(), H::integer()}}
| {incSize, {W::integer(), H::integer()}}.

setSizeHints(This, MinW, MinH, Option::[Option]) -> ok

Types:

```
This = wxWindow()
MinW = integer()
MinH = integer()
Option = {maxW, integer()} | {maxH, integer()} | {incW, integer()} |
{incH, integer()}
```

See **external documentation**.

setSizer(This, Sizer) -> ok

Types:

```
This = wxWindow()
Sizer = wxSizer() (see module wxSizer)
```

Equivalent to *setSizer(This, Sizer, [])*.

setSizer(This, Sizer, Option::[Option]) -> ok

Types:

```
This = wxWindow()
Sizer = wxSizer() (see module wxSizer)
Option = {deleteOld, boolean()}
```

See **external documentation**.

setSizerAndFit(This, Sizer) -> ok

Types:

This = wxWindow()
Sizer = wxSizer() (see module wxSizer)

Equivalent to *setSizerAndFit(This, Sizer, [])*.

setSizerAndFit(This, Sizer, Option::[Option]) -> ok

Types:

This = wxWindow()
Sizer = wxSizer() (see module wxSizer)
Option = {deleteOld, boolean()}

See **external documentation**.

setThemeEnabled(This, EnableTheme) -> ok

Types:

This = wxWindow()
EnableTheme = boolean()

See **external documentation**.

setToolTip(This, Tip) -> ok

Types:

This = wxWindow()
Tip = chardata() (see module unicode)

See **external documentation**.
Also:
setToolTip(This, Tip) -> ok when
This::wxWindow(), Tip::wxToolTip:wxToolTip().

setVirtualSize(This, Size) -> ok

Types:

This = wxWindow()
Size = {W::integer(), H::integer()}

See **external documentation**.

setVirtualSize(This, X, Y) -> ok

Types:

This = wxWindow()
X = integer()
Y = integer()

See **external documentation**.

setVirtualSizeHints(This, MinSize) -> ok

Types:

```
This = wxWindow()
MinSize = {W::integer(), H::integer()}
```

Equivalent to *setVirtualSizeHints(This, MinSize, [])*.

setVirtualSizeHints(This, MinW, MinH) -> ok

Types:

```
This = wxWindow()
MinW = integer()
MinH = integer()
```

See **external documentation**.
Also:
setVirtualSizeHints(This, MinSize, [Option]) -> ok when
This::wxWindow(), MinSize::{W::integer(), H::integer()},
Option :: {maxSize, {W::integer(), H::integer()}}.

setVirtualSizeHints(This, MinW, MinH, Option::[Option]) -> ok

Types:

```
This = wxWindow()
MinW = integer()
MinH = integer()
Option = {maxW, integer()} | {maxH, integer()}
```

See **external documentation**.

setWindowStyle(This, Style) -> ok

Types:

```
This = wxWindow()
Style = integer()
```

See **external documentation**.

setWindowStyleFlag(This, Style) -> ok

Types:

```
This = wxWindow()
Style = integer()
```

See **external documentation**.

setWindowVariant(This, Variant) -> ok

Types:

```
This = wxWindow()
Variant = wx_enum() (see module wx)
```

See **external documentation**.
Variant = ?wxWINDOW_VARIANT_NORMAL | ?wxWINDOW_VARIANT_SMALL | ?wxWINDOW_VARIANT_MINI | ?wxWINDOW_VARIANT_LARGE | ?wxWINDOW_VARIANT_MAX

shouldInheritColours(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

show(This) -> boolean()

Types:

This = wxWindow()

Equivalent to *show(This, [])*.

show(This, Option::[Option]) -> boolean()

Types:

This = wxWindow()

Option = {show, boolean()}

See **external documentation**.

thaw(This) -> ok

Types:

This = wxWindow()

See **external documentation**.

transferDataFromWindow(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

transferDataToWindow(This) -> boolean()

Types:

This = wxWindow()

See **external documentation**.

update(This) -> ok

Types:

This = wxWindow()

See **external documentation**.

updateWindowUI(This) -> ok

Types:

This = wxWindow()

Equivalent to *updateWindowUI(This, [])*.

updateWindowUI(This, Option::[Option]) -> ok

Types:

```
This = wxWindow()
Option = {flags, integer()}
```

See **external documentation**.

validate(This) -> boolean()

Types:

```
This = wxWindow()
```

See **external documentation**.

warpPointer(This, X, Y) -> ok

Types:

```
This = wxWindow()
X = integer()
Y = integer()
```

See **external documentation**.

destroy(This::wxWindow()) -> ok

Destroys this object, do not use object again

wxWindowCreateEvent

Erlang module

See external documentation: **wxWindowCreateEvent**.

Use *wxEvtHandler:connect/3* with EventType:

create

See also the message variant *#wxWindowCreate{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxWindowCreateEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxWindowDC

Erlang module

See external documentation: **wxWindowDC**.

This class is derived (and can use functions) from:
wxDC

DATA TYPES

`wxWindowDC()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxWindowDC()`

See **external documentation**.

`new(Win) -> wxWindowDC()`

Types:

`Win = wxWindow() (see module wxWindow)`

See **external documentation**.

`destroy(This::wxWindowDC()) -> ok`

Destroys this object, do not use object again

wxWindowDestroyEvent

Erlang module

See external documentation: **wxWindowDestroyEvent**.

Use *wxEvtHandler:connect/3* with EventType:

destroy

See also the message variant *#wxWindowDestroy{}* event record type.

This class is derived (and can use functions) from:
wxCommandEvent
wxEvent

DATA TYPES

`wxWindowDestroyEvent()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

wxXmlResource

Erlang module

See external documentation: **wxXmlResource**.

DATA TYPES

`wxXmlResource()`

An object reference, The representation is internal and can be changed without notice. It can't be used for comparsion stored on disc or distributed for use on other nodes.

Exports

`new() -> wxXmlResource()`

Equivalent to *new([])*.

`new(Option::[Option]) -> wxXmlResource()`

Types:

```
Option = {flags, integer()} | {domain, chardata() (see module unicode)}
```

See **external documentation**.

`new(Filemask, Option::[Option]) -> wxXmlResource()`

Types:

```
Filemask = chardata() (see module unicode)
Option = {flags, integer()} | {domain, chardata() (see module unicode)}
```

See **external documentation**.

`attachUnknownControl(This, Name, Control) -> boolean()`

Types:

```
This = wxXmlResource()
Name = chardata() (see module unicode)
Control = wxWindow() (see module wxWindow)
```

Equivalent to *attachUnknownControl(This, Name, Control, [])*.

`attachUnknownControl(This, Name, Control, Option::[Option]) -> boolean()`

Types:

```
This = wxXmlResource()
Name = chardata() (see module unicode)
Control = wxWindow() (see module wxWindow)
Option = {parent, wxWindow() (see module wxWindow)}
```

See **external documentation**.

clearHandlers(This) -> ok

Types:

This = wxXmlResource()

See **external documentation**.

compareVersion(This, Major, Minor, Release, Revision) -> integer()

Types:

This = wxXmlResource()
Major = integer()
Minor = integer()
Release = integer()
Revision = integer()

See **external documentation**.

get() -> wxXmlResource()

See **external documentation**.

getFlags(This) -> integer()

Types:

This = wxXmlResource()

See **external documentation**.

getVersion(This) -> integer()

Types:

This = wxXmlResource()

See **external documentation**.

getXRCID(Str_id) -> integer()

Types:

Str_id = [chardata() (see module unicode)]

Equivalent to *getXRCID(Str_id, [])*.

getXRCID(Str_id, Option::[Option]) -> integer()

Types:

Str_id = [chardata() (see module unicode)]
Option = {value_if_not_found, integer()}

See **external documentation**.

initAllHandlers(This) -> ok

Types:

This = wxXmlResource()

See **external documentation**.

load(This, Filemask) -> boolean()

Types:

This = wxXmlResource()
Filemask = chardata() (see module unicode)

See **external documentation**.

loadBitmap(This, Name) -> wxBitmap() (see module wxBitmap)

Types:

This = wxXmlResource()
Name = chardata() (see module unicode)

See **external documentation**.

loadDialog(This, Parent, Name) -> wxDialog() (see module wxDialog)

Types:

This = wxXmlResource()
Parent = wxWindow() (see module wxWindow)
Name = chardata() (see module unicode)

See **external documentation**.

loadDialog(This, Dlg, Parent, Name) -> boolean()

Types:

This = wxXmlResource()
Dlg = wxDialog() (see module wxDialog)
Parent = wxWindow() (see module wxWindow)
Name = chardata() (see module unicode)

See **external documentation**.

loadFrame(This, Parent, Name) -> wxFrame() (see module wxFrame)

Types:

This = wxXmlResource()
Parent = wxWindow() (see module wxWindow)
Name = chardata() (see module unicode)

See **external documentation**.

loadFrame(This, Frame, Parent, Name) -> boolean()

Types:

This = wxXmlResource()
Frame = wxFrame() (see module wxFrame)
Parent = wxWindow() (see module wxWindow)
Name = chardata() (see module unicode)

See **external documentation**.

loadIcon(This, Name) -> wxIcon() (see module wxIcon)

Types:

This = wxXmlResource()
Name = chardata() (see module unicode)

See **external documentation**.

loadMenu(This, Name) -> wxMenu() (see module wxMenu)

Types:

This = wxXmlResource()
Name = chardata() (see module unicode)

See **external documentation**.

loadMenuBar(This, Name) -> wxMenuBar() (see module wxMenuBar)

Types:

This = wxXmlResource()
Name = chardata() (see module unicode)

See **external documentation**.

loadMenuBar(This, Parent, Name) -> wxMenuBar() (see module wxMenuBar)

Types:

This = wxXmlResource()
Parent = wxWindow() (see module wxWindow)
Name = chardata() (see module unicode)

See **external documentation**.

loadPanel(This, Parent, Name) -> wxPanel() (see module wxPanel)

Types:

This = wxXmlResource()
Parent = wxWindow() (see module wxWindow)
Name = chardata() (see module unicode)

See **external documentation**.

loadPanel(This, Panel, Parent, Name) -> boolean()

Types:

This = wxXmlResource()
Panel = wxPanel() (see module wxPanel)
Parent = wxWindow() (see module wxWindow)
Name = chardata() (see module unicode)

See **external documentation**.

loadToolBar(This, Parent, Name) -> wxToolBar() (see module wxToolBar)

Types:

This = wxXmlResource()

```
Parent = wxWindow() (see module wxWindow)
Name = chardata() (see module unicode)
```

See **external documentation**.

set(Res) -> wxXmlResource()

Types:

```
Res = wxXmlResource()
```

See **external documentation**.

setFlags(This, Flags) -> ok

Types:

```
This = wxXmlResource()
Flags = integer()
```

See **external documentation**.

unload(This, Filename) -> boolean()

Types:

```
This = wxXmlResource()
Filename = chardata() (see module unicode)
```

See **external documentation**.

xrcctrl(Window::wxWindow() (see module wxWindow), Name::string(), Type::atom()) -> wxObject() (see module wx)

Looks up a control with Name in a window created with XML resources. You can use it to set/get values from controls. The object is type casted to *Type*. Example:
Xrc = wxXmlResource:get(),
Dlg = wxDialog:new(),
true = wxXmlResource:loadDialog(Xrc, Dlg, Frame, "controls_dialog"),
LCtrl = xrcctrl(Dlg, "controls_listctrl", wxListCtrl),
wxListCtrl:insertColumn(LCtrl, 0, "Name", [{width, 200}]),

destroy(This::wxXmlResource()) -> ok

Destroys this object, do not use object again

wx_misc

Erlang module

See external documentation: **Misc**.

Exports

getKeyState(Key) -> boolean()

Types:

Key = wx_enum() (see module wx)

See **external documentation**.
Key = ?WXK_BACK | ?WXK_TAB | ?WXK_RETURN | ?WXK_ESCAPE | ?WXK_SPACE | ?WXK_DELETE | ?WXK_START | ?WXK_LBUTTON | ?WXK_RBUTTON | ?WXK_CANCEL | ?WXK_MBUTTON | ?WXK_CLEAR | ?WXK_SHIFT | ?WXK_ALT | ?WXK_CONTROL | ?WXK_MENU | ?WXK_PAUSE | ?WXK_CAPITAL | ?WXK_END | ?WXK_HOME | ?WXK_LEFT | ?WXK_UP | ?WXK_RIGHT | ?WXK_DOWN | ?WXK_SELECT | ?WXK_PRINT | ?WXK_EXECUTE | ?WXK_SNAPSHOT | ?WXK_INSERT | ?WXK_HELP | ?WXK_NUMPAD0 | ?WXK_NUMPAD1 | ?WXK_NUMPAD2 | ?WXK_NUMPAD3 | ?WXK_NUMPAD4 | ?WXK_NUMPAD5 | ?WXK_NUMPAD6 | ?WXK_NUMPAD7 | ?WXK_NUMPAD8 | ?WXK_NUMPAD9 | ?WXK_MULTIPLY | ?WXK_ADD | ?WXK_SEPARATOR | ?WXK_SUBTRACT | ?WXK_DECIMAL | ?WXK_DIVIDE | ?WXK_F1 | ?WXK_F2 | ?WXK_F3 | ?WXK_F4 | ?WXK_F5 | ?WXK_F6 | ?WXK_F7 | ?WXK_F8 | ?WXK_F9 | ?WXK_F10 | ?WXK_F11 | ?WXK_F12 | ?WXK_F13 | ?WXK_F14 | ?WXK_F15 | ?WXK_F16 | ?WXK_F17 | ?WXK_F18 | ?WXK_F19 | ?WXK_F20 | ?WXK_F21 | ?WXK_F22 | ?WXK_F23 | ?WXK_F24 | ?WXK_NUMLOCK | ?WXK_SCROLL | ?WXK_PAGEUP | ?WXK_PAGEDOWN | ?WXK_NUMPAD_SPACE | ?WXK_NUMPAD_TAB | ?WXK_NUMPAD_ENTER | ?WXK_NUMPAD_F1 | ?WXK_NUMPAD_F2 | ?WXK_NUMPAD_F3 | ?WXK_NUMPAD_F4 | ?WXK_NUMPAD_HOME | ?WXK_NUMPAD_LEFT | ?WXK_NUMPAD_UP | ?WXK_NUMPAD_RIGHT | ?WXK_NUMPAD_DOWN | ?WXK_NUMPAD_PAGEUP | ?WXK_NUMPAD_PAGEDOWN | ?WXK_NUMPAD_END | ?WXK_NUMPAD_BEGIN | ?WXK_NUMPAD_INSERT | ?WXK_NUMPAD_DELETE | ?WXK_NUMPAD_EQUAL | ?WXK_NUMPAD_MULTIPLY | ?WXK_NUMPAD_ADD | ?WXK_NUMPAD_SEPARATOR | ?WXK_NUMPAD_SUBTRACT | ?WXK_NUMPAD_DECIMAL | ?WXK_NUMPAD_DIVIDE | ?WXK_WINDOWS_LEFT | ?WXK_WINDOWS_RIGHT | ?WXK_WINDOWS_MENU | ?WXK_COMMAND | ?WXK_SPECIAL1 | ?WXK_SPECIAL2 | ?WXK_SPECIAL3 | ?WXK_SPECIAL4 | ?WXK_SPECIAL5 | ?WXK_SPECIAL6 | ?WXK_SPECIAL7 | ?WXK_SPECIAL8 | ?WXK_SPECIAL9 | ?WXK_SPECIAL10 | ?WXK_SPECIAL11 | ?WXK_SPECIAL12 | ?WXK_SPECIAL13 | ?WXK_SPECIAL14 | ?WXK_SPECIAL15 | ?WXK_SPECIAL16 | ?WXK_SPECIAL17 | ?WXK_SPECIAL18 | ?WXK_SPECIAL19 | ?WXK_SPECIAL20

getMousePosition() -> {X::integer(), Y::integer()}

See **external documentation**.

getMouseState() -> wx_wxMouseState() (see module wx)

See **external documentation**.

setDetectableAutoRepeat(Flag) -> boolean()

Types:

Flag = boolean()

See **external documentation**.

bell() -> ok

See **external documentation**.

findMenuItemId(Frame, MenuString, ItemString) -> integer()

Types:

Frame = wxFrame() (see module wxFrame)
MenuString = chardata() (see module unicode)
ItemString = chardata() (see module unicode)

See **external documentation**.

genericFindWindowAtPoint(Pt) -> wxWindow() (see module wxWindow)

Types:

Pt = {X::integer(), Y::integer()}

See **external documentation**.

findWindowAtPoint(Pt) -> wxWindow() (see module wxWindow)

Types:

Pt = {X::integer(), Y::integer()}

See **external documentation**.

beginBusyCursor() -> ok

Equivalent to *beginBusyCursor([])*.

beginBusyCursor(Option::[Option]) -> ok

Types:

Option = {cursor, wxCursor() (see module wxCursor)}

See **external documentation**.

endBusyCursor() -> ok

See **external documentation**.

isBusy() -> boolean()

See **external documentation**.

shutdown(WFlags) -> boolean()

Types:

WFlags = wx_enum() (see module wx)

See **external documentation**.
WFlags = ?wxSHUTDOWN_POWEROFF | ?wxSHUTDOWN_REBOOT

shell() -> boolean()

Equivalent to *shell([])*.

shell(Option::[Option]) -> boolean()

Types:

Option = {command, chardata() (see module unicode)}

See **external documentation**.

launchDefaultBrowser(Url) -> boolean()

Types:

Url = chardata() (see module unicode)

Equivalent to *launchDefaultBrowser(Url, [])*.

launchDefaultBrowser(Url, Option::[Option]) -> boolean()

Types:

Url = chardata() (see module unicode)

Option = {flags, integer()}

See **external documentation**.

getEmailAddress() -> charlist() (see module unicode)

See **external documentation**.

getUserId() -> charlist() (see module unicode)

See **external documentation**.

getHomeDir() -> charlist() (see module unicode)

See **external documentation**.

newId() -> integer()

See **external documentation**.

registerId(Id) -> ok

Types:

Id = integer()

See **external documentation**.

getCurrentId() -> integer()

See **external documentation**.

getOsDescription() -> charlist() (see module unicode)

See **external documentation**.

isPlatformLittleEndian() -> boolean()

See **external documentation**.

isPlatform64Bit() -> boolean()

See **external documentation**.

glu

Erlang module

A part of the standard OpenGL Utility api. See **www.opengl.org**

Booleans are represented by integers 0 and 1.

DATA TYPES

```
enum() = non_neg_integer()
```

See wx/include/gl.hrl or glu.hrl

```
matrix() = {float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float(), float(), float(), float(),
float()}
mem() = binary() | tuple()
```

Memory block

```
vertex() = {float(), float(), float()}
```

Exports

tesselate(Normal, Vs::[Vs]) -> {Triangles, VertexPos}

Types:

Normal = vertex()
Vs = vertex()
Triangles = [integer()]
VertexPos = binary()

General purpose polygon triangulation. The first argument is the normal and the second a list of vertex positions. Returned is a list of indecies of the vertices and a binary (64bit native float) containing an array of vertex positions, it starts with the vertices in Vs and may contain newly created vertices in the end.

build1DMipmapLevels(Target, InternalFormat, Width, Format, Type, Level, Base, Max, Data) -> integer()

Types:

Target = enum()
InternalFormat = integer()
Width = integer()
Format = enum()
Type = enum()
Level = integer()
Base = integer()
Max = integer()
Data = binary()

Builds a subset of one-dimensional mipmap levels

`glu:build1DMipmapLevels` builds a subset of prefiltered one-dimensional texture maps of decreasing resolutions called a mipmap. This is used for the antialiasing of texture mapped primitives.

A return value of zero indicates success, otherwise a GLU error code is returned (see *glu:errorString/1*).

A series of mipmap levels from `Base` to `Max` is built by decimating `Data` in half until size 1*1 is reached. At each level, each texel in the halved mipmap level is an average of the corresponding two texels in the larger mipmap level. *gl:texImage1D/8* is called to load these mipmap levels from `Base` to `Max` . If `Max` is larger than the highest mipmap level for the texture of the specified size, then a GLU error code is returned (see *glu:errorString/1*) and nothing is loaded.

For example, if `Level` is 2 and `Width` is 16, the following levels are possible: 16*1, 8*1, 4*1, 2*1, 1*1. These correspond to levels 2 through 6 respectively. If `Base` is 3 and `Max` is 5, then only mipmap levels 8*1, 4*1 and 2*1 are loaded. However, if `Max` is 7, then an error is returned and nothing is loaded since `Max` is larger than the highest mipmap level which is, in this case, 6.

The highest mipmap level can be derived from the formula log 2(width*2 level).

See the *gl:texImage1D/8* reference page for a description of the acceptable values for `Type` parameter. See the *gl:drawPixels/5* reference page for a description of the acceptable values for `Level` parameter.

See **external** documentation.

`build1DMipmaps(Target, InternalFormat, Width, Format, Type, Data) -> integer()`

Types:

```
Target = enum()
InternalFormat = integer()
Width = integer()
Format = enum()
Type = enum()
Data = binary()
```

Builds a one-dimensional mipmap

`glu:build1DMipmaps` builds a series of prefiltered one-dimensional texture maps of decreasing resolutions called a mipmap. This is used for the antialiasing of texture mapped primitives.

A return value of zero indicates success, otherwise a GLU error code is returned (see *glu:errorString/1*).

Initially, the `Width` of `Data` is checked to see if it is a power of 2. If not, a copy of `Data` is scaled up or down to the nearest power of 2. (If `Width` is exactly between powers of 2, then the copy of `Data` will scale upwards.) This copy will be used for subsequent mipmapping operations described below. For example, if `Width` is 57, then a copy of `Data` will scale up to 64 before mipmapping takes place.

Then, proxy textures (see *gl:texImage1D/8*) are used to determine if the implementation can fit the requested texture. If not, `Width` is continually halved until it fits.

Next, a series of mipmap levels is built by decimating a copy of `Data` in half until size 1*1 is reached. At each level, each texel in the halved mipmap level is an average of the corresponding two texels in the larger mipmap level.

gl:texImage1D/8 is called to load each of these mipmap levels. Level 0 is a copy of `Data` . The highest level is (log 2)(width). For example, if `Width` is 64 and the implementation can store a texture of this size, the following mipmap levels are built: 64*1, 32*1, 16*1, 8*1, 4*1, 2*1, and 1*1. These correspond to levels 0 through 6, respectively.

See the *gl:texImage1D/8* reference page for a description of the acceptable values for the `Type` parameter. See the *gl:drawPixels/5* reference page for a description of the acceptable values for the `Data` parameter.

See **external** documentation.

build2DMipmapLevels(Target, InternalFormat, Width, Height, Format, Type, Level, Base, Max, Data) -> integer()

Types:

Target = enum()
InternalFormat = integer()
Width = integer()
Height = integer()
Format = enum()
Type = enum()
Level = integer()
Base = integer()
Max = integer()
Data = binary()

Builds a subset of two-dimensional mipmap levels

`glu:build2DMipmapLevels` builds a subset of prefiltered two-dimensional texture maps of decreasing resolutions called a mipmap. This is used for the antialiasing of texture mapped primitives.

A return value of zero indicates success, otherwise a GLU error code is returned (see *glu:errorString/1*).

A series of mipmap levels from `Base` to `Max` is built by decimating `Data` in half along both dimensions until size 1*1 is reached. At each level, each texel in the halved mipmap level is an average of the corresponding four texels in the larger mipmap level. (In the case of rectangular images, the decimation will ultimately reach an N*1 or 1*N configuration. Here, two texels are averaged instead.) *gl:texImage2D/9* is called to load these mipmap levels from `Base` to `Max` . If `Max` is larger than the highest mipmap level for the texture of the specified size, then a GLU error code is returned (see *glu:errorString/1*) and nothing is loaded.

For example, if `Level` is 2 and `Width` is 16 and `Height` is 8, the following levels are possible: 16*8, 8*4, 4*2, 2*1, 1*1. These correspond to levels 2 through 6 respectively. If `Base` is 3 and `Max` is 5, then only mipmap levels 8*4, 4*2, and 2*1 are loaded. However, if `Max` is 7, then an error is returned and nothing is loaded since `Max` is larger than the highest mipmap level which is, in this case, 6.

The highest mipmap level can be derived from the formula log 2(max(width height)*2 level).

See the *gl:texImage1D/8* reference page for a description of the acceptable values for `Format` parameter. See the *gl:drawPixels/5* reference page for a description of the acceptable values for `Type` parameter.

See **external** documentation.

build2DMipmaps(Target, InternalFormat, Width, Height, Format, Type, Data) -> integer()

Types:

Target = enum()
InternalFormat = integer()
Width = integer()
Height = integer()
Format = enum()
Type = enum()
Data = binary()

Builds a two-dimensional mipmap

`glu:build2DMipmaps` builds a series of prefiltered two-dimensional texture maps of decreasing resolutions called a mipmap. This is used for the antialiasing of texture-mapped primitives.

A return value of zero indicates success, otherwise a GLU error code is returned (see *glu:errorString/1*).

Initially, the `Width` and `Height` of `Data` are checked to see if they are a power of 2. If not, a copy of `Data` (not `Data`), is scaled up or down to the nearest power of 2. This copy will be used for subsequent mipmapping operations described below. (If `Width` or `Height` is exactly between powers of 2, then the copy of `Data` will scale upwards.) For example, if `Width` is 57 and `Height` is 23, then a copy of `Data` will scale up to 64 in `Width` and down to 16 in depth, before mipmapping takes place.

Then, proxy textures (see *gl:texImage2D/9*) are used to determine if the implementation can fit the requested texture. If not, both dimensions are continually halved until it fits. (If the OpenGL version is (<= 1.0, both maximum texture dimensions are clamped to the value returned by *gl:getBooleanv/1* with the argument `?GLU_MAX_TEXTURE_SIZE` .)

Next, a series of mipmap levels is built by decimating a copy of `Data` in half along both dimensions until size 1*1 is reached. At each level, each texel in the halved mipmap level is an average of the corresponding four texels in the larger mipmap level. (In the case of rectangular images, the decimation will ultimately reach an N*1 or 1*N configuration. Here, two texels are averaged instead.)

gl:texImage2D/9 is called to load each of these mipmap levels. Level 0 is a copy of `Data` . The highest level is (log 2)(max(width height)). For example, if `Width` is 64 and `Height` is 16 and the implementation can store a texture of this size, the following mipmap levels are built: 64*16, 32*8, 16*4, 8*2, 4*1, 2*1, and 1*1 These correspond to levels 0 through 6, respectively.

See the *gl:texImage1D/8* reference page for a description of the acceptable values for `Format` parameter. See the *gl:drawPixels/5* reference page for a description of the acceptable values for `Type` parameter.

See **external** documentation.

`build3DMipmapLevels(Target, InternalFormat, Width, Height, Depth, Format, Type, Level, Base, Max, Data) -> integer()`

Types:

```
Target = enum()
InternalFormat = integer()
Width = integer()
Height = integer()
Depth = integer()
Format = enum()
Type = enum()
Level = integer()
Base = integer()
Max = integer()
Data = binary()
```

Builds a subset of three-dimensional mipmap levels

`glu:build3DMipmapLevels` builds a subset of prefiltered three-dimensional texture maps of decreasing resolutions called a mipmap. This is used for the antialiasing of texture mapped primitives.

A return value of zero indicates success, otherwise a GLU error code is returned (see *glu:errorString/1*).

A series of mipmap levels from `Base` to `Max` is built by decimating `Data` in half along both dimensions until size 1*1*1 is reached. At each level, each texel in the halved mipmap level is an average of the corresponding eight texels in the larger mipmap level. (If exactly one of the dimensions is 1, four texels are averaged. If exactly two of the dimensions are 1, two texels are averaged.) *gl:texImage3D/10* is called to load these mipmap levels from `Base` to

Max . If Max is larger than the highest mipmap level for the texture of the specified size, then a GLU error code is returned (see *glu:errorString/1*) and nothing is loaded.

For example, if Level is 2 and Width is 16, Height is 8 and Depth is 4, the following levels are possible: 16*8*4, 8*4*2, 4*2*1, 2*1*1, 1*1*1. These correspond to levels 2 through 6 respectively. If Base is 3 and Max is 5, then only mipmap levels 8*4*2, 4*2*1, and 2*1*1 are loaded. However, if Max is 7, then an error is returned and nothing is loaded, since Max is larger than the highest mipmap level which is, in this case, 6.

The highest mipmap level can be derived from the formula log 2(max(width height depth)*2 level).

See the *gl:texImage1D/8* reference page for a description of the acceptable values for Format parameter. See the *gl:drawPixels/5* reference page for a description of the acceptable values for Type parameter.

See **external** documentation.

build3DMipmaps(Target, InternalFormat, Width, Height, Depth, Format, Type, Data) -> integer()

Types:

```
Target = enum()
InternalFormat = integer()
Width = integer()
Height = integer()
Depth = integer()
Format = enum()
Type = enum()
Data = binary()
```

Builds a three-dimensional mipmap

glu:build3DMipmaps builds a series of prefiltered three-dimensional texture maps of decreasing resolutions called a mipmap. This is used for the antialiasing of texture-mapped primitives.

A return value of zero indicates success, otherwise a GLU error code is returned (see *glu:errorString/1*).

Initially, the Width , Height and Depth of Data are checked to see if they are a power of 2. If not, a copy of Data is made and scaled up or down to the nearest power of 2. (If Width , Height , or Depth is exactly between powers of 2, then the copy of Data will scale upwards.) This copy will be used for subsequent mipmapping operations described below. For example, if Width is 57, Height is 23, and Depth is 24, then a copy of Data will scale up to 64 in width, down to 16 in height, and up to 32 in depth before mipmapping takes place.

Then, proxy textures (see *gl:texImage3D/10*) are used to determine if the implementation can fit the requested texture. If not, all three dimensions are continually halved until it fits.

Next, a series of mipmap levels is built by decimating a copy of Data in half along all three dimensions until size 1*1*1 is reached. At each level, each texel in the halved mipmap level is an average of the corresponding eight texels in the larger mipmap level. (If exactly one of the dimensions is 1, four texels are averaged. If exactly two of the dimensions are 1, two texels are averaged.)

gl:texImage3D/10 is called to load each of these mipmap levels. Level 0 is a copy of Data . The highest level is (log 2)(max(width height depth)). For example, if Width is 64, Height is 16, and Depth is 32, and the implementation can store a texture of this size, the following mipmap levels are built: 64*16*32, 32*8*16, 16*4*8, 8*2*4, 4*1*2, 2*1*1, and 1*1*1. These correspond to levels 0 through 6, respectively.

See the *gl:texImage1D/8* reference page for a description of the acceptable values for Format parameter. See the *gl:drawPixels/5* reference page for a description of the acceptable values for Type parameter.

See **external** documentation.

checkExtension(ExtName, ExtString) -> 0 | 1

Types:

ExtName = string()

ExtString = string()

Determines if an extension name is supported

`glu:checkExtension` returns `?GLU_TRUE` if `ExtName` is supported otherwise `?GLU_FALSE` is returned.

This is used to check for the presence for OpenGL, GLU, or GLX extension names by passing the extension strings returned by *gl:getString/1* , *glu:getString/1* , see `glXGetClientString` , see `glXQueryExtensionsString`, or see `glXQueryServerString`, respectively, as `ExtString` .

See **external** documentation.

cylinder(Quad, Base, Top, Height, Slices, Stacks) -> ok

Types:

Quad = integer()

Base = float()

Top = float()

Height = float()

Slices = integer()

Stacks = integer()

Draw a cylinder

`glu:cylinder` draws a cylinder oriented along the `z` axis. The base of the cylinder is placed at `z` = 0 and the top at z= height. Like a sphere, a cylinder is subdivided around the `z` axis into slices and along the `z` axis into stacks.

Note that if `Top` is set to 0.0, this routine generates a cone.

If the orientation is set to `?GLU_OUTSIDE` (with *glu:quadricOrientation/2*), then any generated normals point away from the `z` axis. Otherwise, they point toward the `z` axis.

If texturing is turned on (with *glu:quadricTexture/2*), then texture coordinates are generated so that `t` ranges linearly from 0.0 at `z` = 0 to 1.0 at `z` = `Height` , and `s` ranges from 0.0 at the +`y` axis, to 0.25 at the +`x` axis, to 0.5 at the -`y` axis, to 0.75 at the -`x` axis, and back to 1.0 at the +`y` axis.

See **external** documentation.

deleteQuadric(Quad) -> ok

Types:

Quad = integer()

Destroy a quadrics object

`glu:deleteQuadric` destroys the quadrics object (created with *glu:newQuadric/0*) and frees any memory it uses. Once `glu:deleteQuadric` has been called, `Quad` cannot be used again.

See **external** documentation.

disk(Quad, Inner, Outer, Slices, Loops) -> ok

Types:

Quad = integer()

Inner = float()

```
Outer = float()
Slices = integer()
Loops = integer()
```

Draw a disk

`glu:disk` renders a disk on the `z` = 0 plane. The disk has a radius of `Outer` and contains a concentric circular hole with a radius of `Inner` . If `Inner` is 0, then no hole is generated. The disk is subdivided around the `z` axis into slices (like pizza slices) and also about the `z` axis into rings (as specified by `Slices` and `Loops` , respectively).

With respect to orientation, the +`z` side of the disk is considered to be `outside` (see *glu:quadricOrientation/2*). This means that if the orientation is set to `?GLU_OUTSIDE` , then any normals generated point along the +`z` axis. Otherwise, they point along the -`z` axis.

If texturing has been turned on (with *glu:quadricTexture/2*), texture coordinates are generated linearly such that where r= outer, the value at (`r`, 0, 0) is (1, 0.5), at (0, `r`, 0) it is (0.5, 1), at (-`r`, 0, 0) it is (0, 0.5), and at (0, -`r`, 0) it is (0.5, 0).

See **external** documentation.

errorString(Error) -> string()

Types:

```
Error = enum()
```

Produce an error string from a GL or GLU error code

`glu:errorString` produces an error string from a GL or GLU error code. The string is in ISO Latin 1 format. For example, `glu:errorString`(`?GLU_OUT_OF_MEMORY`) returns the string `out of memory`.

The standard GLU error codes are `?GLU_INVALID_ENUM`, `?GLU_INVALID_VALUE`, and `?GLU_OUT_OF_MEMORY`. Certain other GLU functions can return specialized error codes through callbacks. See the *gl:getError/0* reference page for the list of GL error codes.

See **external** documentation.

getString(Name) -> string()

Types:

```
Name = enum()
```

Return a string describing the GLU version or GLU extensions

`glu:getString` returns a pointer to a static string describing the GLU version or the GLU extensions that are supported.

The version number is one of the following forms:

`major_number.minor_numbermajor_number.minor_number.release_number`.

The version string is of the following form:

`version number<space>vendor-specific information`

Vendor-specific information is optional. Its format and contents depend on the implementation.

The standard GLU contains a basic set of features and capabilities. If a company or group of companies wish to support other features, these may be included as extensions to the GLU. If `Name` is `?GLU_EXTENSIONS`, then `glu:getString` returns a space-separated list of names of supported GLU extensions. (Extension names never contain spaces.)

All strings are null-terminated.

See **external** documentation.

lookAt(EyeX, EyeY, EyeZ, CenterX, CenterY, CenterZ, UpX, UpY, UpZ) -> ok

Types:

EyeX = float()
EyeY = float()
EyeZ = float()
CenterX = float()
CenterY = float()
CenterZ = float()
UpX = float()
UpY = float()
UpZ = float()

Define a viewing transformation

`glu:lookAt` creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an `UP` vector.

The matrix maps the reference point to the negative `z` axis and the eye point to the origin. When a typical projection matrix is used, the center of the scene therefore maps to the center of the viewport. Similarly, the direction described by the `UP` vector projected onto the viewing plane is mapped to the positive `y` axis so that it points upward in the viewport. The `UP` vector must not be parallel to the line of sight from the eye point to the reference point.

Let

F=(centerX-eyeX centerY-eyeY centerZ-eyeZ)

Let `UP` be the vector (upX upY upZ).

Then normalize as follows: f= F/(||F||)

UP"= UP/(||UP||)

Finally, let s= f*UP", and u= s*f.

M is then constructed as follows: M=(s[0] s[1] s[2] 0 u[0] u[1] u[2] 0-f[0]-f[1]-f[2] 0 0 0 0 1)

and `glu:lookAt` is equivalent to glMultMatrixf(M); glTranslated(-eyex, -eyey, -eyez);

See **external** documentation.

newQuadric() -> integer()

Create a quadrics object

`glu:newQuadric` creates and returns a pointer to a new quadrics object. This object must be referred to when calling quadrics rendering and control functions. A return value of 0 means that there is not enough memory to allocate the object.

See **external** documentation.

ortho2D(Left, Right, Bottom, Top) -> ok

Types:

Left = float()
Right = float()
Bottom = float()
Top = float()

Define a 2D orthographic projection matrix

glu:ortho2D sets up a two-dimensional orthographic viewing region. This is equivalent to calling *gl:ortho/6* with near= -1 and far= 1.

See **external** documentation.

partialDisk(Quad, Inner, Outer, Slices, Loops, Start, Sweep) -> ok

Types:

Quad = integer()
Inner = float()
Outer = float()
Slices = integer()
Loops = integer()
Start = float()
Sweep = float()

Draw an arc of a disk

glu:partialDisk renders a partial disk on the z= 0 plane. A partial disk is similar to a full disk, except that only the subset of the disk from Start through Start + Sweep is included (where 0 degrees is along the +f2yf axis, 90 degrees along the +x axis, 180 degrees along the -y axis, and 270 degrees along the -x axis).

The partial disk has a radius of Outer and contains a concentric circular hole with a radius of Inner . If Inner is 0, then no hole is generated. The partial disk is subdivided around the z axis into slices (like pizza slices) and also about the z axis into rings (as specified by Slices and Loops , respectively).

With respect to orientation, the +z side of the partial disk is considered to be outside (see *glu:quadricOrientation/2*). This means that if the orientation is set to ?GLU_OUTSIDE, then any normals generated point along the +z axis. Otherwise, they point along the -z axis.

If texturing is turned on (with *glu:quadricTexture/2*), texture coordinates are generated linearly such that where r= outer, the value at (r, 0, 0) is (1.0, 0.5), at (0, r, 0) it is (0.5, 1.0), at (-r, 0, 0) it is (0.0, 0.5), and at (0, -r, 0) it is (0.5, 0.0).

See **external** documentation.

perspective(Fovy, Aspect, ZNear, ZFar) -> ok

Types:

Fovy = float()
Aspect = float()
ZNear = float()
ZFar = float()

Set up a perspective projection matrix

glu:perspective specifies a viewing frustum into the world coordinate system. In general, the aspect ratio in glu:perspective should match the aspect ratio of the associated viewport. For example, aspect= 2.0 means the viewer's angle of view is twice as wide in x as it is in y. If the viewport is twice as wide as it is tall, it displays the image without distortion.

The matrix generated by glu:perspective is multipled by the current matrix, just as if *gl:multMatrixd/1* were called with the generated matrix. To load the perspective matrix onto the current matrix stack instead, precede the call to glu:perspective with a call to *gl:loadIdentity/0* .

Given f defined as follows:

f= cotangent(fovy/2) The generated matrix is

(f/aspect 0 0 0 0 f 0 0 0 0(zFar+zNear)/(zNear-zFar)(2*zFar*zNear)/(zNear-zFar) 0 0 -1 0)

See **external** documentation.

pickMatrix(X, Y, DelX, DelY, Viewport) -> ok

Types:

X = float()
Y = float()
DelX = float()
DelY = float()
Viewport = {integer(), integer(), integer(), integer()}

Define a picking region

`glu:pickMatrix` creates a projection matrix that can be used to restrict drawing to a small region of the viewport. This is typically useful to determine what objects are being drawn near the cursor. Use `glu:pickMatrix` to restrict drawing to a small region around the cursor. Then, enter selection mode (with *gl:renderMode/1*) and rerender the scene. All primitives that would have been drawn near the cursor are identified and stored in the selection buffer.

The matrix created by `glu:pickMatrix` is multiplied by the current matrix just as if *gl:multMatrixd/1* is called with the generated matrix. To effectively use the generated pick matrix for picking, first call *gl:loadIdentity/0* to load an identity matrix onto the perspective matrix stack. Then call `glu:pickMatrix`, and, finally, call a command (such as *glu:perspective/4*) to multiply the perspective matrix by the pick matrix.

When using `glu:pickMatrix` to pick NURBS, be careful to turn off the NURBS property `?GLU_AUTO_LOAD_MATRIX`. If `?GLU_AUTO_LOAD_MATRIX` is not turned off, then any NURBS surface rendered is subdivided differently with the pick matrix than the way it was subdivided without the pick matrix.

See **external** documentation.

project(ObjX, ObjY, ObjZ, Model, Proj, View) -> {integer(), WinX::float(), WinY::float(), WinZ::float()}

Types:

ObjX = float()
ObjY = float()
ObjZ = float()
Model = matrix()
Proj = matrix()
View = {integer(), integer(), integer(), integer()}

Map object coordinates to window coordinates

`glu:project` transforms the specified object coordinates into window coordinates using `Model` , `Proj` , and `View` . The result is stored in `WinX` , `WinY` , and `WinZ` . A return value of `?GLU_TRUE` indicates success, a return value of `?GLU_FALSE` indicates failure.

To compute the coordinates, let v=(objX objY objZ 1.0) represented as a matrix with 4 rows and 1 column. Then `glu:project` computes v'' as follows:

v''= P*M*v

where P is the current projection matrix `Proj` and M is the current modelview matrix `Model` (both represented as 4*4 matrices in column-major order).

The window coordinates are then computed as follows:

winX= view(0)+view(2)*(v''(0)+1)/2

winY= view(1)+view(3)*(v''(1)+1)/2

winZ=(v''(2)+1)/2

See **external** documentation.

quadricDrawStyle(Quad, Draw) -> ok

Types:

```
Quad = integer()
Draw = enum()
```

Specify the draw style desired for quadrics

`glu:quadricDrawStyle` specifies the draw style for quadrics rendered with `Quad` . The legal values are as follows:

`?GLU_FILL`: Quadrics are rendered with polygon primitives. The polygons are drawn in a counterclockwise fashion with respect to their normals (as defined with *glu:quadricOrientation/2*).

`?GLU_LINE`: Quadrics are rendered as a set of lines.

`?GLU_SILHOUETTE`: Quadrics are rendered as a set of lines, except that edges separating coplanar faces will not be drawn.

`?GLU_POINT`: Quadrics are rendered as a set of points.

See **external** documentation.

quadricNormals(Quad, Normal) -> ok

Types:

```
Quad = integer()
Normal = enum()
```

Specify what kind of normals are desired for quadrics

`glu:quadricNormals` specifies what kind of normals are desired for quadrics rendered with `Quad` . The legal values are as follows:

`?GLU_NONE`: No normals are generated.

`?GLU_FLAT`: One normal is generated for every facet of a quadric.

`?GLU_SMOOTH`: One normal is generated for every vertex of a quadric. This is the initial value.

See **external** documentation.

quadricOrientation(Quad, Orientation) -> ok

Types:

```
Quad = integer()
Orientation = enum()
```

Specify inside/outside orientation for quadrics

`glu:quadricOrientation` specifies what kind of orientation is desired for quadrics rendered with `Quad` . The `Orientation` values are as follows:

`?GLU_OUTSIDE`: Quadrics are drawn with normals pointing outward (the initial value).

`?GLU_INSIDE`: Quadrics are drawn with normals pointing inward.

Note that the interpretation of `outward` and `inward` depends on the quadric being drawn.

See **external** documentation.

quadricTexture(Quad, Texture) -> ok

Types:

Quad = integer()
Texture = 0 | 1

Specify if texturing is desired for quadrics

`glu:quadricTexture` specifies if texture coordinates should be generated for quadrics rendered with `Quad` . If the value of `Texture` is `?GLU_TRUE`, then texture coordinates are generated, and if `Texture` is `?GLU_FALSE`, they are not. The initial value is `?GLU_FALSE`.

The manner in which texture coordinates are generated depends upon the specific quadric rendered.

See **external** documentation.

scaleImage(Format, WIn, HIn, TypeIn, DataIn, WOut, HOut, TypeOut, DataOut) -> integer()

Types:

Format = enum()
WIn = integer()
HIn = integer()
TypeIn = enum()
DataIn = binary()
WOut = integer()
HOut = integer()
TypeOut = enum()
DataOut = mem()

Scale an image to an arbitrary size

`glu:scaleImage` scales a pixel image using the appropriate pixel store modes to unpack data from the source image and pack data into the destination image.

When shrinking an image, `glu:scaleImage` uses a box filter to sample the source image and create pixels for the destination image. When magnifying an image, the pixels from the source image are linearly interpolated to create the destination image.

A return value of zero indicates success, otherwise a GLU error code is returned (see *glu:errorString/1*).

See the *gl:readPixels/7* reference page for a description of the acceptable values for the `Format` , `TypeIn` , and `TypeOut` parameters.

See **external** documentation.

sphere(Quad, Radius, Slices, Stacks) -> ok

Types:

Quad = integer()
Radius = float()
Slices = integer()
Stacks = integer()

Draw a sphere

`glu:sphere` draws a sphere of the given radius centered around the origin. The sphere is subdivided around the `z` axis into slices and along the `z` axis into stacks (similar to lines of longitude and latitude).

If the orientation is set to `?GLU_OUTSIDE` (with *glu:quadricOrientation/2*), then any normals generated point away from the center of the sphere. Otherwise, they point toward the center of the sphere.

If texturing is turned on (with *glu:quadricTexture/2*), then texture coordinates are generated so that `t` ranges from 0.0 at z=-radius to 1.0 at z= radius (`t` increases linearly along longitudinal lines), and `s` ranges from 0.0 at the +`y` axis, to 0.25 at the +`x` axis, to 0.5 at the -`y` axis, to 0.75 at the -`x` axis, and back to 1.0 at the +`y` axis.

See **external** documentation.

`unProject(WinX, WinY, WinZ, Model, Proj, View) -> {integer(), ObjX::float(), ObjY::float(), ObjZ::float()}`

Types:

```
WinX = float()
WinY = float()
WinZ = float()
Model = matrix()
Proj = matrix()
View = {integer(), integer(), integer(), integer()}
```

Map window coordinates to object coordinates

`glu:unProject` maps the specified window coordinates into object coordinates using `Model` , `Proj` , and `View` . The result is stored in `ObjX` , `ObjY` , and `ObjZ` . A return value of `?GLU_TRUE` indicates success; a return value of `?GLU_FALSE` indicates failure.

To compute the coordinates (objX objY objZ), `glu:unProject` multiplies the normalized device coordinates by the inverse of `Model` * `Proj` as follows:

(objX objY objZ W)= INV(P M) ((2(winX-view[0]))/(view[2])-1(2(winY-view[1]))/(view[3])-1 2(winZ)-1 1) INV denotes matrix inversion. W is an unused variable, included for consistent matrix notation.

See **external** documentation.

`unProject4(WinX, WinY, WinZ, ClipW, Model, Proj, View, NearVal, FarVal) -> {integer(), ObjX::float(), ObjY::float(), ObjZ::float(), ObjW::float()}`

Types:

```
WinX = float()
WinY = float()
WinZ = float()
ClipW = float()
Model = matrix()
Proj = matrix()
View = {integer(), integer(), integer(), integer()}
NearVal = float()
FarVal = float()
```

See *unProject/6*

gl

Erlang module

Standard OpenGL api. See **www.opengl.org**

Booleans are represented by integers 0 and 1.

DATA TYPES

`clamp() = float()`

0.0..1.0

`enum() = non_neg_integer()`

See wx/include/gl.hrl

```
matrix() = {float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float(), float(), float(), float(),
float()}
mem() = binary() | tuple()
```

Memory block

`offset() = non_neg_integer()`

Offset in memory block

Exports

`clearIndex(C) -> ok`

Types:

`C = float()`

Specify the clear value for the color index buffers

`gl:clearIndex` specifies the index used by *gl:clear/1* to clear the color index buffers. `C` is not clamped. Rather, `C` is converted to a fixed-point value with unspecified precision to the right of the binary point. The integer part of this value is then masked with 2 m-1, where m is the number of bits in a color index stored in the frame buffer.

See **external** documentation.

`clearColor(Red, Green, Blue, Alpha) -> ok`

Types:

`Red = clamp()`
`Green = clamp()`
`Blue = clamp()`
`Alpha = clamp()`

Specify clear values for the color buffers

`gl:clearColor` specifies the red, green, blue, and alpha values used by *gl:clear/1* to clear the color buffers. Values specified by `gl:clearColor` are clamped to the range [0 1].

See **external** documentation.

clear(Mask) -> ok

Types:

Mask = integer()

Clear buffers to preset values

`gl:clear` sets the bitplane area of the window to values previously selected by `gl:clearColor` , `gl:clearDepth`, and `gl:clearStencil`. Multiple color buffers can be cleared simultaneously by selecting more than one buffer at a time using *gl:drawBuffer/1* .

The pixel ownership test, the scissor test, dithering, and the buffer writemasks affect the operation of `gl:clear`. The scissor box bounds the cleared region. Alpha function, blend function, logical operation, stenciling, texture mapping, and depth-buffering are ignored by `gl:clear`.

`gl:clear` takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

The values are as follows:

`?GL_COLOR_BUFFER_BIT`: Indicates the buffers currently enabled for color writing.

`?GL_DEPTH_BUFFER_BIT`: Indicates the depth buffer.

`?GL_STENCIL_BUFFER_BIT`: Indicates the stencil buffer.

The value to which each buffer is cleared depends on the setting of the clear value for that buffer.

See **external** documentation.

indexMask(Mask) -> ok

Types:

Mask = integer()

Control the writing of individual bits in the color index buffers

`gl:indexMask` controls the writing of individual bits in the color index buffers. The least significant n bits of `Mask` , where n is the number of bits in a color index buffer, specify a mask. Where a 1 (one) appears in the mask, it's possible to write to the corresponding bit in the color index buffer (or buffers). Where a 0 (zero) appears, the corresponding bit is write-protected.

This mask is used only in color index mode, and it affects only the buffers currently selected for writing (see *gl:drawBuffer/1*). Initially, all bits are enabled for writing.

See **external** documentation.

colorMask(Red, Green, Blue, Alpha) -> ok

Types:

Red = 0 | 1

Green = 0 | 1

Blue = 0 | 1

Alpha = 0 | 1

Enable and disable writing of frame buffer color components

`gl:colorMask` and `gl:colorMaski` specify whether the individual color components in the frame buffer can or cannot be written. `gl:colorMaski` sets the mask for a specific draw buffer, whereas `gl:colorMask` sets the mask for all draw buffers. If `Red` is `?GL_FALSE`, for example, no change is made to the red component of any pixel in any of the color buffers, regardless of the drawing operation attempted.

Changes to individual bits of components cannot be controlled. Rather, changes are either enabled or disabled for entire color components.

See **external** documentation.

alphaFunc(Func, Ref) -> ok

Types:

```
Func = enum()
Ref = clamp()
```

Specify the alpha test function

The alpha test discards fragments depending on the outcome of a comparison between an incoming fragment's alpha value and a constant reference value. `gl:alphaFunc` specifies the reference value and the comparison function. The comparison is performed only if alpha testing is enabled. By default, it is not enabled. (See *gl:enable/1* and *gl:enable/1* of `?GL_ALPHA_TEST`.)

`Func` and `Ref` specify the conditions under which the pixel is drawn. The incoming alpha value is compared to `Ref` using the function specified by `Func` . If the value passes the comparison, the incoming fragment is drawn if it also passes subsequent stencil and depth buffer tests. If the value fails the comparison, no change is made to the frame buffer at that pixel location. The comparison functions are as follows:

`?GL_NEVER`: Never passes.

`?GL_LESS`: Passes if the incoming alpha value is less than the reference value.

`?GL_EQUAL`: Passes if the incoming alpha value is equal to the reference value.

`?GL_LEQUAL`: Passes if the incoming alpha value is less than or equal to the reference value.

`?GL_GREATER`: Passes if the incoming alpha value is greater than the reference value.

`?GL_NOTEQUAL`: Passes if the incoming alpha value is not equal to the reference value.

`?GL_GEQUAL`: Passes if the incoming alpha value is greater than or equal to the reference value.

`?GL_ALWAYS`: Always passes (initial value).

`gl:alphaFunc` operates on all pixel write operations, including those resulting from the scan conversion of points, lines, polygons, and bitmaps, and from pixel draw and copy operations. `gl:alphaFunc` does not affect screen clear operations.

See **external** documentation.

blendFunc(Sfactor, Dfactor) -> ok

Types:

```
Sfactor = enum()
Dfactor = enum()
```

Specify pixel arithmetic

Pixels can be drawn using a function that blends the incoming (source) RGBA values with the RGBA values that are already in the frame buffer (the destination values). Blending is initially disabled. Use *gl:enable/1* and *gl:enable/1* with argument `?GL_BLEND` to enable and disable blending.

`gl:blendFunc` defines the operation of blending for all draw buffers when it is enabled. `gl:blendFunci` defines the operation of blending for a single draw buffer specified by `Buf` when enabled for that draw buffer. `Sfactor` specifies which method is used to scale the source color components. `Dfactor` specifies which method is used to scale the destination color components. Both parameters must be one of the following symbolic constants: `?GL_ZERO`, `?GL_ONE`, `?`

GL_SRC_COLOR, ?GL_ONE_MINUS_SRC_COLOR , ?GL_DST_COLOR, ?GL_ONE_MINUS_DST_COLOR, ?GL_SRC_ALPHA, ?GL_ONE_MINUS_SRC_ALPHA , ?GL_DST_ALPHA, ?GL_ONE_MINUS_DST_ALPHA, ?GL_CONSTANT_COLOR, ?GL_ONE_MINUS_CONSTANT_COLOR , ?GL_CONSTANT_ALPHA, ?GL_ONE_MINUS_CONSTANT_ALPHA, ?GL_SRC_ALPHA_SATURATE , ?GL_SRC1_COLOR, ?GL_ONE_MINUS_SRC1_COLOR, ?GL_SRC1_ALPHA, and ?GL_ONE_MINUS_SRC1_ALPHA . The possible methods are described in the following table. Each method defines four scale factors, one each for red, green, blue, and alpha. In the table and in subsequent equations, first source, second source and destination color components are referred to as (R s0 G s0 B s0 A s0), (R s1 G s1 B s1 A s1) and (R d G d B d A d), respectively. The color specified by *gl:blendColor/4* is referred to as (R c G c B c A c). They are understood to have integer values between 0 and (k R k G k B k A), where

k c= 2(m c)-1

and (m R m G m B m A) is the number of red, green, blue, and alpha bitplanes.

Source and destination scale factors are referred to as (s R s G s B s A) and (d R d G d B d A). The scale factors described in the table, denoted (f R f G f B f A), represent either source or destination factors. All scale factors have range [0 1].

Parameter(f R f G f B f A)
?GL_ZERO (0 0 0 0)
?GL_ONE(1 1 1 1)
?GL_SRC_COLOR (R s0 k/R G s0 k/G B s0 k/B A s0 k/A)
?GL_ONE_MINUS_SRC_COLOR(1 1 1 1)-(R s0 k/R G s0 k/G B s0 k/B A s0 k/A)
?GL_DST_COLOR (R d k/R G d k/G B d k/B A d k/A)
?GL_ONE_MINUS_DST_COLOR(1 1 1 1)-(R d k/R G d k/G B d k/B A d k/A)
?GL_SRC_ALPHA (A s0 k/A A s0 k/A A s0 k/A A s0 k/A)
?GL_ONE_MINUS_SRC_ALPHA(1 1 1 1)-(A s0 k/A A s0 k/A A s0 k/A A s0 k/A)
?GL_DST_ALPHA (A d k/A A d k/A A d k/A A d k/A)
?GL_ONE_MINUS_DST_ALPHA(1 1 1 1)-(A d k/A A d k/A A d k/A A d k/A)
?GL_CONSTANT_COLOR (R c G c B c A c)
?GL_ONE_MINUS_CONSTANT_COLOR(1 1 1 1)-(R c G c B c A c)
?GL_CONSTANT_ALPHA(A c A c A c A c)
?GL_ONE_MINUS_CONSTANT_ALPHA (1 1 1 1)-(A c A c A c A c)
?GL_SRC_ALPHA_SATURATE(i i i 1)
?GL_SRC1_COLOR (R s1 k/R G s1 k/G B s1 k/B A s1 k/A)
?GL_ONE_MINUS_SRC1_COLOR(1 1 1 1)-(R s1 k/R G s1 k/G B s1 k/B A s1 k/A)
?GL_SRC1_ALPHA (A s1 k/A A s1 k/A A s1 k/A A s1 k/A)
?GL_ONE_MINUS_SRC1_ALPHA(1 1 1 1)-(A s1 k/A A s1 k/A A s1 k/A A s1 k/A)

In the table,

i= min(A s k A-A d) k/A

To determine the blended RGBA values of a pixel, the system uses the following equations:

R d= min(k R R s s R+R d d R) G d= min(k G G s s G+G d d G) B d= min(k B B s s B+B d d B) A d= min(k A A s s A+A d d A)

Despite the apparent precision of the above equations, blending arithmetic is not exactly specified, because blending operates with imprecise integer color values. However, a blend factor that should be equal to 1 is guaranteed not to modify its multiplicand, and a blend factor equal to 0 reduces its multiplicand to 0. For example, when Sfactor is ?GL_SRC_ALPHA , Dfactor is ?GL_ONE_MINUS_SRC_ALPHA, and A s is equal to k A, the equations reduce to simple replacement:

R d= R s G d= G s B d= B s A d= A s

See **external** documentation.

logicOp(Opcode) -> ok

Types:

Opcode = enum()

Specify a logical pixel operation for rendering

`gl:logicOp` specifies a logical operation that, when enabled, is applied between the incoming RGBA color and the RGBA color at the corresponding location in the frame buffer. To enable or disable the logical operation, call *gl:enable/1* and *gl:enable/1* using the symbolic constant `?GL_COLOR_LOGIC_OP`. The initial value is disabled.

`Opcode` Resulting Operation
`?GL_CLEAR` 0
`?GL_SET` 1
`?GL_COPY` s
`?GL_COPY_INVERTED` ~s
`?GL_NOOP` d
`?GL_INVERT` ~d
`?GL_AND` s & d
`?GL_NAND` ~(s & d)
`?GL_OR` s | d
`?GL_NOR` ~(s | d)
`?GL_XOR` s ^ d
`?GL_EQUIV` ~(s ^ d)
`?GL_AND_REVERSE` s & ~d
`?GL_AND_INVERTED` ~s & d
`?GL_OR_REVERSE` s | ~d
`?GL_OR_INVERTED` ~s | d

`Opcode` is a symbolic constant chosen from the list above. In the explanation of the logical operations, `s` represents the incoming color and `d` represents the color in the frame buffer. Standard C-language operators are used. As these bitwise operators suggest, the logical operation is applied independently to each bit pair of the source and destination colors.

See **external** documentation.

cullFace(Mode) -> ok

Types:

Mode = enum()

Specify whether front- or back-facing facets can be culled

`gl:cullFace` specifies whether front- or back-facing facets are culled (as specified by `mode`) when facet culling is enabled. Facet culling is initially disabled. To enable and disable facet culling, call the *gl:enable/1* and *gl:enable/1* commands with the argument `?GL_CULL_FACE`. Facets include triangles, quadrilaterals, polygons, and rectangles.

gl:frontFace/1 specifies which of the clockwise and counterclockwise facets are front-facing and back-facing. See *gl:frontFace/1* .

See **external** documentation.

frontFace(Mode) -> ok

Types:

Mode = enum()

Define front- and back-facing polygons

In a scene composed entirely of opaque closed surfaces, back-facing polygons are never visible. Eliminating these invisible polygons has the obvious benefit of speeding up the rendering of the image. To enable and disable elimination of back-facing polygons, call *gl:enable/1* and *gl:enable/1* with argument `?GL_CULL_FACE`.

The projection of a polygon to window coordinates is said to have clockwise winding if an imaginary object following the path from its first vertex, its second vertex, and so on, to its last vertex, and finally back to its first vertex, moves in a clockwise direction about the interior of the polygon. The polygon's winding is said to be counterclockwise if the imaginary object following the same path moves in a counterclockwise direction about the interior of the polygon. `gl:frontFace` specifies whether polygons with clockwise winding in window coordinates, or counterclockwise winding in window coordinates, are taken to be front-facing. Passing `?GL_CCW` to `Mode` selects counterclockwise polygons as front-facing; `?GL_CW` selects clockwise polygons as front-facing. By default, counterclockwise polygons are taken to be front-facing.

See **external** documentation.

`pointSize(Size) -> ok`

Types:

`Size = float()`

Specify the diameter of rasterized points

`gl:pointSize` specifies the rasterized diameter of points. If point size mode is disabled (see *gl:enable/1* with parameter `?GL_PROGRAM_POINT_SIZE`), this value will be used to rasterize points. Otherwise, the value written to the shading language built-in variable gl_PointSize will be used.

See **external** documentation.

`lineWidth(Width) -> ok`

Types:

`Width = float()`

Specify the width of rasterized lines

`gl:lineWidth` specifies the rasterized width of both aliased and antialiased lines. Using a line width other than 1 has different effects, depending on whether line antialiasing is enabled. To enable and disable line antialiasing, call *gl:enable/1* and *gl:enable/1* with argument `?GL_LINE_SMOOTH`. Line antialiasing is initially disabled.

If line antialiasing is disabled, the actual width is determined by rounding the supplied width to the nearest integer. (If the rounding results in the value 0, it is as if the line width were 1.) If |Δ x|>=|Δ y|, `i` pixels are filled in each column that is rasterized, where `i` is the rounded value of `Width` . Otherwise, `i` pixels are filled in each row that is rasterized.

If antialiasing is enabled, line rasterization produces a fragment for each pixel square that intersects the region lying within the rectangle having width equal to the current line width, length equal to the actual length of the line, and centered on the mathematical line segment. The coverage value for each fragment is the window coordinate area of the intersection of the rectangular region with the corresponding pixel square. This value is saved and used in the final rasterization step.

Not all widths can be supported when line antialiasing is enabled. If an unsupported width is requested, the nearest supported width is used. Only width 1 is guaranteed to be supported; others depend on the implementation. Likewise, there is a range for aliased line widths as well. To query the range of supported widths and the size difference between supported widths within the range, call *gl:getBooleanv/1* with arguments `?GL_ALIASED_LINE_WIDTH_RANGE` , `?GL_SMOOTH_LINE_WIDTH_RANGE`, and `?GL_SMOOTH_LINE_WIDTH_GRANULARITY`.

See **external** documentation.

lineStipple(Factor, Pattern) -> ok

Types:

Factor = integer()

Pattern = integer()

Specify the line stipple pattern

Line stippling masks out certain fragments produced by rasterization; those fragments will not be drawn. The masking is achieved by using three parameters: the 16-bit line stipple pattern `Pattern` , the repeat count `Factor` , and an integer stipple counter s.

Counter s is reset to 0 whenever *gl:'begin'/1* is called and before each line segment of a *gl:'begin'/1* (`?GL_LINES`)/ *gl:'begin'/1* sequence is generated. It is incremented after each fragment of a unit width aliased line segment is generated or after each i fragments of an i width line segment are generated. The i fragments associated with count s are masked out if

`Pattern` bit (s/factor)% 16

is 0, otherwise these fragments are sent to the frame buffer. Bit zero of `Pattern` is the least significant bit.

Antialiased lines are treated as a sequence of 1*width rectangles for purposes of stippling. Whether rectangle s is rasterized or not depends on the fragment rule described for aliased lines, counting rectangles rather than groups of fragments.

To enable and disable line stippling, call *gl:enable/1* and *gl:enable/1* with argument `?GL_LINE_STIPPLE`. When enabled, the line stipple pattern is applied as described above. When disabled, it is as if the pattern were all 1's. Initially, line stippling is disabled.

See **external** documentation.

polygonMode(Face, Mode) -> ok

Types:

Face = enum()

Mode = enum()

Select a polygon rasterization mode

`gl:polygonMode` controls the interpretation of polygons for rasterization. `Face` describes which polygons `Mode` applies to: both front and back-facing polygons (`?GL_FRONT_AND_BACK`). The polygon mode affects only the final rasterization of polygons. In particular, a polygon's vertices are lit and the polygon is clipped and possibly culled before these modes are applied.

Three modes are defined and can be specified in `Mode` :

`?GL_POINT`: Polygon vertices that are marked as the start of a boundary edge are drawn as points. Point attributes such as `?GL_POINT_SIZE` and `?GL_POINT_SMOOTH` control the rasterization of the points. Polygon rasterization attributes other than `?GL_POLYGON_MODE` have no effect.

`?GL_LINE`: Boundary edges of the polygon are drawn as line segments. Line attributes such as `?GL_LINE_WIDTH` and `?GL_LINE_SMOOTH` control the rasterization of the lines. Polygon rasterization attributes other than `?GL_POLYGON_MODE` have no effect.

`?GL_FILL`: The interior of the polygon is filled. Polygon attributes such as `?GL_POLYGON_SMOOTH` control the rasterization of the polygon.

See **external** documentation.

polygonOffset(Factor, Units) -> ok

Types:

Factor = float()
Units = float()

Set the scale and units used to calculate depth values

When ?GL_POLYGON_OFFSET_FILL, ?GL_POLYGON_OFFSET_LINE, or ?GL_POLYGON_OFFSET_POINT is enabled, each fragment's depth value will be offset after it is interpolated from the depth values of the appropriate vertices. The value of the offset is factor*DZ+r*units, where DZ is a measurement of the change in depth relative to the screen area of the polygon, and r is the smallest value that is guaranteed to produce a resolvable offset for a given implementation. The offset is added before the depth test is performed and before the value is written into the depth buffer.

gl:polygonOffset is useful for rendering hidden-line images, for applying decals to surfaces, and for rendering solids with highlighted edges.

See **external** documentation.

polygonStipple(Mask) -> ok

Types:

Mask = binary()

Set the polygon stippling pattern

Polygon stippling, like line stippling (see *gl:lineStipple/2*), masks out certain fragments produced by rasterization, creating a pattern. Stippling is independent of polygon antialiasing.

Pattern is a pointer to a 32*32 stipple pattern that is stored in memory just like the pixel data supplied to a *gl:drawPixels/5* call with height and width both equal to 32, a pixel format of ?GL_COLOR_INDEX, and data type of ?GL_BITMAP . That is, the stipple pattern is represented as a 32*32 array of 1-bit color indices packed in unsigned bytes. *gl:pixelStoref/2* parameters like ?GL_UNPACK_SWAP_BYTES and ?GL_UNPACK_LSB_FIRST affect the assembling of the bits into a stipple pattern. Pixel transfer operations (shift, offset, pixel map) are not applied to the stipple image, however.

If a non-zero named buffer object is bound to the ?GL_PIXEL_UNPACK_BUFFER target (see *gl:bindBuffer/2*) while a stipple pattern is specified, Pattern is treated as a byte offset into the buffer object's data store.

To enable and disable polygon stippling, call *gl:enable/1* and *gl:enable/1* with argument ?GL_POLYGON_STIPPLE. Polygon stippling is initially disabled. If it's enabled, a rasterized polygon fragment with window coordinates x w and y w is sent to the next stage of the GL if and only if the (x w% 32)th bit in the (y w% 32)th row of the stipple pattern is 1 (one). When polygon stippling is disabled, it is as if the stipple pattern consists of all 1's.

See **external** documentation.

getPolygonStipple() -> binary()

Return the polygon stipple pattern

gl:getPolygonStipple returns to Pattern a 32*32 polygon stipple pattern. The pattern is packed into memory as if *gl:readPixels/7* with both height and width of 32, type of ?GL_BITMAP, and format of ?GL_COLOR_INDEX were called, and the stipple pattern were stored in an internal 32*32 color index buffer. Unlike *gl:readPixels/7* , however, pixel transfer operations (shift, offset, pixel map) are not applied to the returned stipple image.

If a non-zero named buffer object is bound to the ?GL_PIXEL_PACK_BUFFER target (see *gl:bindBuffer/2*) while a polygon stipple pattern is requested, Pattern is treated as a byte offset into the buffer object's data store.

See **external** documentation.

edgeFlag(Flag) -> ok

Types:

Flag = 0 | 1

Flag edges as either boundary or nonboundary

Each vertex of a polygon, separate triangle, or separate quadrilateral specified between a *gl:'begin'/1* / *gl:'begin'/1* pair is marked as the start of either a boundary or nonboundary edge. If the current edge flag is true when the vertex is specified, the vertex is marked as the start of a boundary edge. Otherwise, the vertex is marked as the start of a nonboundary edge. `gl:edgeFlag` sets the edge flag bit to `?GL_TRUE` if `Flag` is `?GL_TRUE` and to `?GL_FALSE` otherwise.

The vertices of connected triangles and connected quadrilaterals are always marked as boundary, regardless of the value of the edge flag.

Boundary and nonboundary edge flags on vertices are significant only if `?GL_POLYGON_MODE` is set to `?GL_POINT` or `?GL_LINE`. See *gl:polygonMode/2* .

See **external** documentation.

edgeFlagv(Flag) -> ok

Types:

Flag = {Flag::0 | 1}

Equivalent to *edgeFlag(Flag)*.

scissor(X, Y, Width, Height) -> ok

Types:

X = integer()
Y = integer()
Width = integer()
Height = integer()

Define the scissor box

`gl:scissor` defines a rectangle, called the scissor box, in window coordinates. The first two arguments, `X` and `Y` , specify the lower left corner of the box. `Width` and `Height` specify the width and height of the box.

To enable and disable the scissor test, call *gl:enable/1* and *gl:enable/1* with argument `?GL_SCISSOR_TEST`. The test is initially disabled. While the test is enabled, only pixels that lie within the scissor box can be modified by drawing commands. Window coordinates have integer values at the shared corners of frame buffer pixels. glScissor(0,0,1,1) allows modification of only the lower left pixel in the window, and glScissor(0,0,0,0) doesn't allow modification of any pixels in the window.

When the scissor test is disabled, it is as though the scissor box includes the entire window.

See **external** documentation.

clipPlane(Plane, Equation) -> ok

Types:

Plane = enum()
Equation = {float(), float(), float(), float()}

Specify a plane against which all geometry is clipped

Geometry is always clipped against the boundaries of a six-plane frustum in x, y , and z. gl:clipPlane allows the specification of additional planes, not necessarily perpendicular to the x, y, or z axis, against which all geometry is clipped. To determine the maximum number of additional clipping planes, call *gl:getBooleanv/1* with argument ?GL_MAX_CLIP_PLANES. All implementations support at least six such clipping planes. Because the resulting clipping region is the intersection of the defined half-spaces, it is always convex.

gl:clipPlane specifies a half-space using a four-component plane equation. When gl:clipPlane is called, Equation is transformed by the inverse of the modelview matrix and stored in the resulting eye coordinates. Subsequent changes to the modelview matrix have no effect on the stored plane-equation components. If the dot product of the eye coordinates of a vertex with the stored plane equation components is positive or zero, the vertex is in with respect to that clipping plane. Otherwise, it is out.

To enable and disable clipping planes, call *gl:enable/1* and *gl:enable/1* with the argument ?GL_CLIP_PLANEi, where i is the plane number.

All clipping planes are initially defined as (0, 0, 0, 0) in eye coordinates and are disabled.

See **external** documentation.

getClipPlane(Plane) -> {float(), float(), float(), float()}

Types:

Plane = enum()

Return the coefficients of the specified clipping plane

gl:getClipPlane returns in Equation the four coefficients of the plane equation for Plane .

See **external** documentation.

drawBuffer(Mode) -> ok

Types:

Mode = enum()

Specify which color buffers are to be drawn into

When colors are written to the frame buffer, they are written into the color buffers specified by gl:drawBuffer. The specifications are as follows:

?GL_NONE: No color buffers are written.

?GL_FRONT_LEFT: Only the front left color buffer is written.

?GL_FRONT_RIGHT: Only the front right color buffer is written.

?GL_BACK_LEFT: Only the back left color buffer is written.

?GL_BACK_RIGHT: Only the back right color buffer is written.

?GL_FRONT: Only the front left and front right color buffers are written. If there is no front right color buffer, only the front left color buffer is written.

?GL_BACK: Only the back left and back right color buffers are written. If there is no back right color buffer, only the back left color buffer is written.

?GL_LEFT: Only the front left and back left color buffers are written. If there is no back left color buffer, only the front left color buffer is written.

?GL_RIGHT: Only the front right and back right color buffers are written. If there is no back right color buffer, only the front right color buffer is written.

?GL_FRONT_AND_BACK: All the front and back color buffers (front left, front right, back left, back right) are written. If there are no back color buffers, only the front left and front right color buffers are written. If there are no right color

buffers, only the front left and back left color buffers are written. If there are no right or back color buffers, only the front left color buffer is written.

If more than one color buffer is selected for drawing, then blending or logical operations are computed and applied independently for each color buffer and can produce different results in each buffer.

Monoscopic contexts include only `left` buffers, and stereoscopic contexts include both `left` and `right` buffers. Likewise, single-buffered contexts include only `front` buffers, and double-buffered contexts include both `front` and `back` buffers. The context is selected at GL initialization.

See **external** documentation.

readBuffer(Mode) -> ok

Types:

Mode = enum()

Select a color buffer source for pixels

`gl:readBuffer` specifies a color buffer as the source for subsequent *gl:readPixels/7* , *gl:copyTexImage1D/7* , *gl:copyTexImage2D/8* , *gl:copyTexSubImage1D/6* , *gl:copyTexSubImage2D/8* , and *gl:copyTexSubImage3D/9* commands. `Mode` accepts one of twelve or more predefined values. In a fully configured system, `?GL_FRONT`, `?GL_LEFT`, and `?GL_FRONT_LEFT` all name the front left buffer, `?GL_FRONT_RIGHT` and `?GL_RIGHT` name the front right buffer, and `?GL_BACK_LEFT` and `?GL_BACK` name the back left buffer. Further more, the constants `?GL_COLOR_ATTACHMENTi` may be used to indicate the `i`th color attachment where `i` ranges from zero to the value of `?GL_MAX_COLOR_ATTACHMENTS` minus one.

Nonstereo double-buffered configurations have only a front left and a back left buffer. Single-buffered configurations have a front left and a front right buffer if stereo, and only a front left buffer if nonstereo. It is an error to specify a nonexistent buffer to `gl:readBuffer` .

`Mode` is initially `?GL_FRONT` in single-buffered configurations and `?GL_BACK` in double-buffered configurations.

See **external** documentation.

enable(Cap) -> ok

Types:

Cap = enum()

Enable or disable server-side GL capabilities

`gl:enable` and *gl:enable/1* enable and disable various capabilities. Use *gl:isEnabled/1* or *gl:getBooleanv/1* to determine the current setting of any capability. The initial value for each capability with the exception of `?GL_DITHER` and `?GL_MULTISAMPLE` is `?GL_FALSE`. The initial value for `?GL_DITHER` and `?GL_MULTISAMPLE` is `?GL_TRUE`.

Both `gl:enable` and *gl:enable/1* take a single argument, `Cap` , which can assume one of the following values:

Some of the GL's capabilities are indexed. `gl:enablei` and `gl:disablei` enable and disable indexed capabilities.

`?GL_BLEND`: If enabled, blend the computed fragment color values with the values in the color buffers. See *gl:blendFunc/2* .

`?GL_CLIP_DISTANCEi`: If enabled, clip geometry against user-defined half space `i`.

`?GL_COLOR_LOGIC_OP`: If enabled, apply the currently selected logical operation to the computed fragment color and color buffer values. See *gl:logicOp/1* .

`?GL_CULL_FACE`: If enabled, cull polygons based on their winding in window coordinates. See *gl:cullFace/1* .

`?GL_DEPTH_CLAMP`: If enabled, the -w c≤ z c≤ w c plane equation is ignored by view volume clipping (effectively, there is no near or far plane clipping). See *gl:depthRange/2* .

?GL_DEPTH_TEST: If enabled, do depth comparisons and update the depth buffer. Note that even if the depth buffer exists and the depth mask is non-zero, the depth buffer is not updated if the depth test is disabled. See *gl:depthFunc/1* and *gl:depthRange/2* .

?GL_DITHER: If enabled, dither color components or indices before they are written to the color buffer.

?GL_FRAMEBUFFER_SRGB: If enabled and the value of ?GL_FRAMEBUFFER_ATTACHMENT_COLOR_ENCODING for the framebuffer attachment corresponding to the destination buffer is ?GL_SRGB, the R, G, and B destination color values (after conversion from fixed-point to floating-point) are considered to be encoded for the sRGB color space and hence are linearized prior to their use in blending.

?GL_LINE_SMOOTH: If enabled, draw lines with correct filtering. Otherwise, draw aliased lines. See *gl:lineWidth/1* .

?GL_MULTISAMPLE: If enabled, use multiple fragment samples in computing the final color of a pixel. See *gl:sampleCoverage/2* .

?GL_POLYGON_OFFSET_FILL: If enabled, and if the polygon is rendered in ?GL_FILL mode, an offset is added to depth values of a polygon's fragments before the depth comparison is performed. See *gl:polygonOffset/2* .

?GL_POLYGON_OFFSET_LINE: If enabled, and if the polygon is rendered in ?GL_LINE mode, an offset is added to depth values of a polygon's fragments before the depth comparison is performed. See *gl:polygonOffset/2* .

?GL_POLYGON_OFFSET_POINT: If enabled, an offset is added to depth values of a polygon's fragments before the depth comparison is performed, if the polygon is rendered in ?GL_POINT mode. See *gl:polygonOffset/2* .

?GL_POLYGON_SMOOTH: If enabled, draw polygons with proper filtering. Otherwise, draw aliased polygons. For correct antialiased polygons, an alpha buffer is needed and the polygons must be sorted front to back.

?GL_PRIMITIVE_RESTART: Enables primitive restarting. If enabled, any one of the draw commands which transfers a set of generic attribute array elements to the GL will restart the primitive when the index of the vertex is equal to the primitive restart index. See *gl:primitiveRestartIndex/1* .

?GL_SAMPLE_ALPHA_TO_COVERAGE: If enabled, compute a temporary coverage value where each bit is determined by the alpha value at the corresponding sample location. The temporary coverage value is then ANDed with the fragment coverage value.

?GL_SAMPLE_ALPHA_TO_ONE: If enabled, each sample alpha value is replaced by the maximum representable alpha value.

?GL_SAMPLE_COVERAGE: If enabled, the fragment's coverage is ANDed with the temporary coverage value. If ?GL_SAMPLE_COVERAGE_INVERT is set to ?GL_TRUE, invert the coverage value. See *gl:sampleCoverage/2* .

?GL_SAMPLE_SHADING: If enabled, the active fragment shader is run once for each covered sample, or at fraction of this rate as determined by the current value of ?GL_MIN_SAMPLE_SHADING_VALUE . See *gl:minSampleShading/1* .

?GL_SAMPLE_MASK: If enabled, the sample coverage mask generated for a fragment during rasterization will be ANDed with the value of ?GL_SAMPLE_MASK_VALUE before shading occurs. See *gl:sampleMaski/2* .

?GL_SCISSOR_TEST: If enabled, discard fragments that are outside the scissor rectangle. See *gl:scissor/4* .

?GL_STENCIL_TEST: If enabled, do stencil testing and update the stencil buffer. See *gl:stencilFunc/3* and *gl:stencilOp/3* .

?GL_TEXTURE_CUBE_MAP_SEAMLESS: If enabled, cubemap textures are sampled such that when linearly sampling from the border between two adjacent faces, texels from both faces are used to generate the final sample value. When disabled, texels from only a single face are used to construct the final sample value.

?GL_PROGRAM_POINT_SIZE: If enabled and a vertex or geometry shader is active, then the derived point size is taken from the (potentially clipped) shader builtin ?gl_PointSize and clamped to the implementation-dependent point size range.

See **external** documentation.

disable(Cap) -> ok

Types:

Cap = enum()

See *enable/1*

isEnabled(Cap) -> 0 | 1

Types:

Cap = enum()

Test whether a capability is enabled

`gl:isEnabled` returns `?GL_TRUE` if `Cap` is an enabled capability and returns `?GL_FALSE` otherwise. Boolean states that are indexed may be tested with `gl:isEnabledi` . For `gl:isEnabledi`, `Index` specifies the index of the capability to test. `Index` must be between zero and the count of indexed capabilities for `Cap` . Initially all capabilities except `?GL_DITHER` are disabled; `?GL_DITHER` is initially enabled.

The following capabilities are accepted for `Cap` :

Constant	See
`?GL_BLEND`	*gl:blendFunc/2* , *gl:logicOp/1*
`?GL_CLIP_DISTANCEi`	*gl:enable/1*
`?GL_COLOR_LOGIC_OP`	*gl:logicOp/1*
`?GL_CULL_FACE`	*gl:cullFace/1*
`?GL_DEPTH_CLAMP`	*gl:enable/1*
`?GL_DEPTH_TEST`	*gl:depthFunc/1* , *gl:depthRange/2*
`?GL_DITHER`	*gl:enable/1*
`?GL_FRAMEBUFFER_SRGB`	*gl:enable/1*
`?GL_LINE_SMOOTH`	*gl:lineWidth/1*
`?GL_MULTISAMPLE`	*gl:sampleCoverage/2*
`?GL_POLYGON_SMOOTH`	*gl:polygonMode/2*
`?GL_POLYGON_OFFSET_FILL`	*gl:polygonOffset/2*
`?GL_POLYGON_OFFSET_LINE`	*gl:polygonOffset/2*
`?GL_POLYGON_OFFSET_POINT`	*gl:polygonOffset/2*
`?GL_PROGRAM_POINT_SIZE`	*gl:enable/1*
`?GL_PRIMITIVE_RESTART`	*gl:enable/1* , *gl:primitiveRestartIndex/1*
`?GL_SAMPLE_ALPHA_TO_COVERAGE`	*gl:sampleCoverage/2*
`?GL_SAMPLE_ALPHA_TO_ONE`	*gl:sampleCoverage/2*
`?GL_SAMPLE_COVERAGE`	*gl:sampleCoverage/2*
`?GL_SAMPLE_MASK`	*gl:enable/1*
`?GL_SCISSOR_TEST`	*gl:scissor/4*
`?GL_STENCIL_TEST`	*gl:stencilFunc/3* , *gl:stencilOp/3*
`?GL_TEXTURE_CUBEMAP_SEAMLESS`	*gl:enable/1*

See **external** documentation.

enableClientState(Cap) -> ok

Types:

Cap = enum()

Enable or disable client-side capability

`gl:enableClientState` and *gl:enableClientState/1* enable or disable individual client-side capabilities. By default, all client-side capabilities are disabled. Both `gl:enableClientState` and *gl:enableClientState/1* take a single argument, `Cap` , which can assume one of the following values:

?GL_COLOR_ARRAY: If enabled, the color array is enabled for writing and used during rendering when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6 gl:multiDrawArrays/3* , or see `glMultiDrawElements` is called. See *gl:colorPointer/4* .

?GL_EDGE_FLAG_ARRAY: If enabled, the edge flag array is enabled for writing and used during rendering when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6 gl:multiDrawArrays/3* , or see `glMultiDrawElements` is called. See *gl:edgeFlagPointer/2* .

?GL_FOG_COORD_ARRAY: If enabled, the fog coordinate array is enabled for writing and used during rendering when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6 gl:multiDrawArrays/3* , or see `glMultiDrawElements` is called. See *gl:fogCoordPointer/3* .

?GL_INDEX_ARRAY: If enabled, the index array is enabled for writing and used during rendering when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6 gl:multiDrawArrays/3* , or see `glMultiDrawElements` is called. See *gl:indexPointer/3* .

?GL_NORMAL_ARRAY: If enabled, the normal array is enabled for writing and used during rendering when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6 gl:multiDrawArrays/3* , or see `glMultiDrawElements` is called. See *gl:normalPointer/3* .

?GL_SECONDARY_COLOR_ARRAY: If enabled, the secondary color array is enabled for writing and used during rendering when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6 gl:multiDrawArrays/3* , or see `glMultiDrawElements` is called. See *gl:colorPointer/4* .

?GL_TEXTURE_COORD_ARRAY: If enabled, the texture coordinate array is enabled for writing and used during rendering when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6 gl:multiDrawArrays/3* , or see `glMultiDrawElements` is called. See *gl:texCoordPointer/4* .

?GL_VERTEX_ARRAY: If enabled, the vertex array is enabled for writing and used during rendering when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6 gl:multiDrawArrays/3* , or see `glMultiDrawElements` is called. See *gl:vertexPointer/4* .

See **external** documentation.

`disableClientState(Cap) -> ok`

Types:

`Cap = enum()`

See *enableClientState/1*

`getBooleanv(Pname) -> [0 | 1]`

Types:

`Pname = enum()`

Return the value or values of a selected parameter

These four commands return values for simple state variables in GL. `Pname` is a symbolic constant indicating the state variable to be returned, and `Params` is a pointer to an array of the indicated type in which to place the returned data.

Type conversion is performed if `Params` has a different type than the state variable value being requested. If `gl:getBooleanv` is called, a floating-point (or integer) value is converted to `?GL_FALSE` if and only if it is 0.0 (or 0). Otherwise, it is converted to `?GL_TRUE`. If `gl:getIntegerv` is called, boolean values are returned as `?GL_TRUE` or `?GL_FALSE`, and most floating-point values are rounded to the nearest integer value. Floating-point colors and normals, however, are returned with a linear mapping that maps 1.0 to the most positive representable integer value and -1.0 to the most negative representable integer value. If `gl:getFloatv` or `gl:getDoublev` is called, boolean values are returned as `?GL_TRUE` or `?GL_FALSE`, and integer values are converted to floating-point values.

The following symbolic constants are accepted by `Pname` :

?GL_ACTIVE_TEXTURE: Params returns a single value indicating the active multitexture unit. The initial value is ?GL_TEXTURE0. See *gl:activeTexture/1* .

?GL_ALIASED_LINE_WIDTH_RANGE: Params returns a pair of values indicating the range of widths supported for aliased lines. See *gl:lineWidth/1* .

?GL_ARRAY_BUFFER_BINDING: Params returns a single value, the name of the buffer object currently bound to the target ?GL_ARRAY_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See *gl:bindBuffer/2* .

?GL_BLEND: Params returns a single boolean value indicating whether blending is enabled. The initial value is ?GL_FALSE. See *gl:blendFunc/2* .

?GL_BLEND_COLOR: Params returns four values, the red, green, blue, and alpha values which are the components of the blend color. See *gl:blendColor/4* .

?GL_BLEND_DST_ALPHA: Params returns one value, the symbolic constant identifying the alpha destination blend function. The initial value is ?GL_ZERO. See *gl:blendFunc/2* and *gl:blendFuncSeparate/4* .

?GL_BLEND_DST_RGB: Params returns one value, the symbolic constant identifying the RGB destination blend function. The initial value is ?GL_ZERO. See *gl:blendFunc/2* and *gl:blendFuncSeparate/4* .

?GL_BLEND_EQUATION_RGB: Params returns one value, a symbolic constant indicating whether the RGB blend equation is ?GL_FUNC_ADD, ?GL_FUNC_SUBTRACT, ?GL_FUNC_REVERSE_SUBTRACT , ?GL_MIN or ?GL_MAX. See *gl:blendEquationSeparate/2* .

?GL_BLEND_EQUATION_ALPHA: Params returns one value, a symbolic constant indicating whether the Alpha blend equation is ?GL_FUNC_ADD, ?GL_FUNC_SUBTRACT , ?GL_FUNC_REVERSE_SUBTRACT, ?GL_MIN or ?GL_MAX. See *gl:blendEquationSeparate/2* .

?GL_BLEND_SRC_ALPHA: Params returns one value, the symbolic constant identifying the alpha source blend function. The initial value is ?GL_ONE. See *gl:blendFunc/2* and *gl:blendFuncSeparate/4* .

?GL_BLEND_SRC_RGB: Params returns one value, the symbolic constant identifying the RGB source blend function. The initial value is ?GL_ONE. See *gl:blendFunc/2* and *gl:blendFuncSeparate/4* .

?GL_COLOR_CLEAR_VALUE: Params returns four values: the red, green, blue, and alpha values used to clear the color buffers. Integer values, if requested, are linearly mapped from the internal floating-point representation such that 1.0 returns the most positive representable integer value, and -1.0 returns the most negative representable integer value. The initial value is (0, 0, 0, 0). See *gl:clearColor/4* .

?GL_COLOR_LOGIC_OP: Params returns a single boolean value indicating whether a fragment's RGBA color values are merged into the framebuffer using a logical operation. The initial value is ?GL_FALSE. See *gl:logicOp/1* .

?GL_COLOR_WRITEMASK: Params returns four boolean values: the red, green, blue, and alpha write enables for the color buffers. The initial value is (?GL_TRUE, ?GL_TRUE, ?GL_TRUE, ?GL_TRUE). See *gl:colorMask/4* .

?GL_COMPRESSED_TEXTURE_FORMATS: Params returns a list of symbolic constants of length ?GL_NUM_COMPRESSED_TEXTURE_FORMATS indicating which compressed texture formats are available. See *gl:compressedTexImage2D/8* .

?GL_CONTEXT_FLAGS: Params returns one value, the flags with which the context was created (such as debugging functionality).

?GL_CULL_FACE: Params returns a single boolean value indicating whether polygon culling is enabled. The initial value is ?GL_FALSE. See *gl:cullFace/1* .

?GL_CURRENT_PROGRAM: Params returns one value, the name of the program object that is currently active, or 0 if no program object is active. See *gl:useProgram/1* .

?GL_DEPTH_CLEAR_VALUE: Params returns one value, the value that is used to clear the depth buffer. Integer values, if requested, are linearly mapped from the internal floating-point representation such that 1.0 returns the most

positive representable integer value, and -1.0 returns the most negative representable integer value. The initial value is 1. See *gl:clearDepth/1* .

?GL_DEPTH_FUNC: Params returns one value, the symbolic constant that indicates the depth comparison function. The initial value is ?GL_LESS. See *gl:depthFunc/1* .

?GL_DEPTH_RANGE: Params returns two values: the near and far mapping limits for the depth buffer. Integer values, if requested, are linearly mapped from the internal floating-point representation such that 1.0 returns the most positive representable integer value, and -1.0 returns the most negative representable integer value. The initial value is (0, 1). See *gl:depthRange/2* .

?GL_DEPTH_TEST: Params returns a single boolean value indicating whether depth testing of fragments is enabled. The initial value is ?GL_FALSE. See *gl:depthFunc/1* and *gl:depthRange/2* .

?GL_DEPTH_WRITEMASK: Params returns a single boolean value indicating if the depth buffer is enabled for writing. The initial value is ?GL_TRUE. See *gl:depthMask/1* .

?GL_DITHER: Params returns a single boolean value indicating whether dithering of fragment colors and indices is enabled. The initial value is ?GL_TRUE.

?GL_DOUBLEBUFFER: Params returns a single boolean value indicating whether double buffering is supported.

?GL_DRAW_BUFFER: Params returns one value, a symbolic constant indicating which buffers are being drawn to. See *gl:drawBuffer/1* . The initial value is ?GL_BACK if there are back buffers, otherwise it is ?GL_FRONT.

?GL_DRAW_BUFFERi: Params returns one value, a symbolic constant indicating which buffers are being drawn to by the corresponding output color. See *gl:drawBuffers/1* . The initial value of ?GL_DRAW_BUFFER0 is ?GL_BACK if there are back buffers, otherwise it is ?GL_FRONT. The initial values of draw buffers for all other output colors is ?GL_NONE.

?GL_DRAW_FRAMEBUFFER_BINDING: Params returns one value, the name of the framebuffer object currently bound to the ?GL_DRAW_FRAMEBUFFER target. If the default framebuffer is bound, this value will be zero. The initial value is zero. See *gl:bindFramebuffer/2* .

?GL_READ_FRAMEBUFFER_BINDING: Params returns one value, the name of the framebuffer object currently bound to the ?GL_READ_FRAMEBUFFER target. If the default framebuffer is bound, this value will be zero. The initial value is zero. See *gl:bindFramebuffer/2* .

?GL_ELEMENT_ARRAY_BUFFER_BINDING: Params returns a single value, the name of the buffer object currently bound to the target ?GL_ELEMENT_ARRAY_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See *gl:bindBuffer/2* .

?GL_FRAGMENT_SHADER_DERIVATIVE_HINT: Params returns one value, a symbolic constant indicating the mode of the derivative accuracy hint for fragment shaders. The initial value is ?GL_DONT_CARE. See *gl:hint/2* .

?GL_IMPLEMENTATION_COLOR_READ_FORMAT: Params returns a single GLenum value indicating the implementation's preferred pixel data format. See *gl:readPixels/7* .

?GL_IMPLEMENTATION_COLOR_READ_TYPE: Params returns a single GLenum value indicating the implementation's preferred pixel data type. See *gl:readPixels/7* .

?GL_LINE_SMOOTH: Params returns a single boolean value indicating whether antialiasing of lines is enabled. The initial value is ?GL_FALSE. See *gl:lineWidth/1* .

?GL_LINE_SMOOTH_HINT: Params returns one value, a symbolic constant indicating the mode of the line antialiasing hint. The initial value is ?GL_DONT_CARE. See *gl:hint/2* .

?GL_LINE_WIDTH: Params returns one value, the line width as specified with *gl:lineWidth/1* . The initial value is 1.

?GL_LAYER_PROVOKING_VERTEX: Params returns one value, the implementation dependent specifc vertex of a primitive that is used to select the rendering layer. If the value returned is equivalent to ?GL_PROVOKING_VERTEX, then the vertex selection follows the convention specified by *gl:provokingVertex/1* . If the value returned is equivalent

to ?GL_FIRST_VERTEX_CONVENTION, then the selection is always taken from the first vertex in the primitive. If the value returned is equivalent to ?GL_LAST_VERTEX_CONVENTION , then the selection is always taken from the last vertex in the primitive. If the value returned is equivalent to ?GL_UNDEFINED_VERTEX, then the selection is not guaranteed to be taken from any specific vertex in the primitive.

?GL_LINE_WIDTH_GRANULARITY: Params returns one value, the width difference between adjacent supported widths for antialiased lines. See *gl:lineWidth/1* .

?GL_LINE_WIDTH_RANGE: Params returns two values: the smallest and largest supported widths for antialiased lines. See *gl:lineWidth/1* .

?GL_LOGIC_OP_MODE: Params returns one value, a symbolic constant indicating the selected logic operation mode. The initial value is ?GL_COPY. See *gl:logicOp/1* .

?GL_MAJOR_VERSION: Params returns one value, the major version number of the OpenGL API supported by the current context.

?GL_MAX_3D_TEXTURE_SIZE: Params returns one value, a rough estimate of the largest 3D texture that the GL can handle. The value must be at least 64. Use ?GL_PROXY_TEXTURE_3D to determine if a texture is too large. See *gl:texImage3D/10* .

?GL_MAX_ARRAY_TEXTURE_LAYERS: Params returns one value. The value indicates the maximum number of layers allowed in an array texture, and must be at least 256. See *gl:texImage2D/9* .

?GL_MAX_CLIP_DISTANCES: Params returns one value, the maximum number of application-defined clipping distances. The value must be at least 8.

?GL_MAX_COLOR_TEXTURE_SAMPLES: Params returns one value, the maximum number of samples in a color multisample texture.

?GL_MAX_COMBINED_ATOMIC_COUNTERS: Params returns a single value, the maximum number of atomic counters available to all active shaders.

?GL_MAX_COMBINED_FRAGMENT_UNIFORM_COMPONENTS: Params returns one value, the number of words for fragment shader uniform variables in all uniform blocks (including default). The value must be at least 1. See *gl:uniform1f/2* .

?GL_MAX_COMBINED_GEOMETRY_UNIFORM_COMPONENTS: Params returns one value, the number of words for geometry shader uniform variables in all uniform blocks (including default). The value must be at least 1. See *gl:uniform1f/2* .

?GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS: Params returns one value, the maximum supported texture image units that can be used to access texture maps from the vertex shader and the fragment processor combined. If both the vertex shader and the fragment processing stage access the same texture image unit, then that counts as using two texture image units against this limit. The value must be at least 48. See *gl:activeTexture/1* .

?GL_MAX_COMBINED_UNIFORM_BLOCKS: Params returns one value, the maximum number of uniform blocks per program. The value must be at least 36. See *gl:uniformBlockBinding/3* .

?GL_MAX_COMBINED_VERTEX_UNIFORM_COMPONENTS: Params returns one value, the number of words for vertex shader uniform variables in all uniform blocks (including default). The value must be at least 1. See *gl:uniform1f/2* .

?GL_MAX_CUBE_MAP_TEXTURE_SIZE: Params returns one value. The value gives a rough estimate of the largest cube-map texture that the GL can handle. The value must be at least 1024. Use ?GL_PROXY_TEXTURE_CUBE_MAP to determine if a texture is too large. See *gl:texImage2D/9* .

?GL_MAX_DEPTH_TEXTURE_SAMPLES: Params returns one value, the maximum number of samples in a multisample depth or depth-stencil texture.

?GL_MAX_DRAW_BUFFERS: Params returns one value, the maximum number of simultaneous outputs that may be written in a fragment shader. The value must be at least 8. See *gl:drawBuffers/1* .

?GL_MAX_DUALSOURCE_DRAW_BUFFERS: Params returns one value, the maximum number of active draw buffers when using dual-source blending. The value must be at least 1. See *gl:blendFunc/2* and *gl:blendFuncSeparate/4* .

?GL_MAX_ELEMENTS_INDICES: Params returns one value, the recommended maximum number of vertex array indices. See *gl:drawRangeElements/6* .

?GL_MAX_ELEMENTS_VERTICES: Params returns one value, the recommended maximum number of vertex array vertices. See *gl:drawRangeElements/6* .

?GL_MAX_FRAGMENT_ATOMIC_COUNTERS: Params returns a single value, the maximum number of atomic counters available to fragment shaders.

?GL_MAX_FRAGMENT_INPUT_COMPONENTS: Params returns one value, the maximum number of components of the inputs read by the fragment shader, which must be at least 128.

?GL_MAX_FRAGMENT_UNIFORM_COMPONENTS: Params returns one value, the maximum number of individual floating-point, integer, or boolean values that can be held in uniform variable storage for a fragment shader. The value must be at least 1024. See *gl:uniform1f/2* .

?GL_MAX_FRAGMENT_UNIFORM_VECTORS: Params returns one value, the maximum number of individual 4-vectors of floating-point, integer, or boolean values that can be held in uniform variable storage for a fragment shader. The value is equal to the value of ?GL_MAX_FRAGMENT_UNIFORM_COMPONENTS divided by 4 and must be at least 256. See *gl:uniform1f/2* .

?GL_MAX_FRAGMENT_UNIFORM_BLOCKS: Params returns one value, the maximum number of uniform blocks per fragment shader. The value must be at least 12. See *gl:uniformBlockBinding/3* .

?GL_MAX_GEOMETRY_ATOMIC_COUNTERS: Params returns a single value, the maximum number of atomic counters available to geometry shaders.

?GL_MAX_GEOMETRY_INPUT_COMPONENTS: Params returns one value, the maximum number of components of inputs read by a geometry shader, which must be at least 64.

?GL_MAX_GEOMETRY_OUTPUT_COMPONENTS: Params returns one value, the maximum number of components of outputs written by a geometry shader, which must be at least 128.

?GL_MAX_GEOMETRY_TEXTURE_IMAGE_UNITS: Params returns one value, the maximum supported texture image units that can be used to access texture maps from the geometry shader. The value must be at least 16. See *gl:activeTexture/1* .

?GL_MAX_GEOMETRY_UNIFORM_BLOCKS: Params returns one value, the maximum number of uniform blocks per geometry shader. The value must be at least 12. See *gl:uniformBlockBinding/3* .

?GL_MAX_GEOMETRY_UNIFORM_COMPONENTS: Params returns one value, the maximum number of individual floating-point, integer, or boolean values that can be held in uniform variable storage for a geometry shader. The value must be at least 1024. See *gl:uniform1f/2* .

?GL_MAX_INTEGER_SAMPLES: Params returns one value, the maximum number of samples supported in integer format multisample buffers.

?GL_MIN_MAP_BUFFER_ALIGNMENT: Params returns one value, the minimum alignment in basic machine units of pointers returned fromsee glMapBuffer and see glMapBufferRange . This value must be a power of two and must be at least 64.

?GL_MAX_PROGRAM_TEXEL_OFFSET: Params returns one value, the maximum texel offset allowed in a texture lookup, which must be at least 7.

?GL_MIN_PROGRAM_TEXEL_OFFSET: Params returns one value, the minimum texel offset allowed in a texture lookup, which must be at most -8.

?GL_MAX_RECTANGLE_TEXTURE_SIZE: Params returns one value. The value gives a rough estimate of the largest rectangular texture that the GL can handle. The value must be at least 1024. Use ?GL_PROXY_RECTANGLE_TEXTURE to determine if a texture is too large. See *gl:texImage2D/9* .

?GL_MAX_RENDERBUFFER_SIZE: Params returns one value. The value indicates the maximum supported size for renderbuffers. See *gl:framebufferRenderbuffer/4* .

?GL_MAX_SAMPLE_MASK_WORDS: Params returns one value, the maximum number of sample mask words.

?GL_MAX_SERVER_WAIT_TIMEOUT: Params returns one value, the maximum *gl:waitSync/3* timeout interval.

?GL_MAX_TESS_CONTROL_ATOMIC_COUNTERS: Params returns a single value, the maximum number of atomic counters available to tessellation control shaders.

?GL_MAX_TESS_EVALUATION_ATOMIC_COUNTERS: Params returns a single value, the maximum number of atomic counters available to tessellation evaluation shaders.

?GL_MAX_TEXTURE_BUFFER_SIZE: Params returns one value. The value gives the maximum number of texels allowed in the texel array of a texture buffer object. Value must be at least 65536.

?GL_MAX_TEXTURE_IMAGE_UNITS: Params returns one value, the maximum supported texture image units that can be used to access texture maps from the fragment shader. The value must be at least 16. See *gl:activeTexture/1* .

?GL_MAX_TEXTURE_LOD_BIAS: Params returns one value, the maximum, absolute value of the texture level-of-detail bias. The value must be at least 2.0.

?GL_MAX_TEXTURE_SIZE: Params returns one value. The value gives a rough estimate of the largest texture that the GL can handle. The value must be at least 1024. Use a proxy texture target such as ?GL_PROXY_TEXTURE_1D or ?GL_PROXY_TEXTURE_2D to determine if a texture is too large. See *gl:texImage1D/8* and *gl:texImage2D/9* .

?GL_MAX_UNIFORM_BUFFER_BINDINGS: Params returns one value, the maximum number of uniform buffer binding points on the context, which must be at least 36.

?GL_MAX_UNIFORM_BLOCK_SIZE: Params returns one value, the maximum size in basic machine units of a uniform block, which must be at least 16384.

?GL_MAX_VARYING_COMPONENTS: Params returns one value, the number components for varying variables, which must be at least 60.

?GL_MAX_VARYING_VECTORS: Params returns one value, the number 4-vectors for varying variables, which is equal to the value of ?GL_MAX_VARYING_COMPONENTS and must be at least 15.

?GL_MAX_VARYING_FLOATS: Params returns one value, the maximum number of interpolators available for processing varying variables used by vertex and fragment shaders. This value represents the number of individual floating-point values that can be interpolated; varying variables declared as vectors, matrices, and arrays will all consume multiple interpolators. The value must be at least 32.

?GL_MAX_VERTEX_ATOMIC_COUNTERS: Params returns a single value, the maximum number of atomic counters available to vertex shaders.

?GL_MAX_VERTEX_ATTRIBS: Params returns one value, the maximum number of 4-component generic vertex attributes accessible to a vertex shader. The value must be at least 16. See *gl:vertexAttrib1d/2* .

?GL_MAX_VERTEX_TEXTURE_IMAGE_UNITS: Params returns one value, the maximum supported texture image units that can be used to access texture maps from the vertex shader. The value may be at least 16. See *gl:activeTexture/1* .

?GL_MAX_VERTEX_UNIFORM_COMPONENTS: Params returns one value, the maximum number of individual floating-point, integer, or boolean values that can be held in uniform variable storage for a vertex shader. The value must be at least 1024. See *gl:uniform1f/2* .

?GL_MAX_VERTEX_UNIFORM_VECTORS: Params returns one value, the maximum number of 4-vectors that may be held in uniform variable storage for the vertex shader. The value of ?GL_MAX_VERTEX_UNIFORM_VECTORS is equal to the value of ?GL_MAX_VERTEX_UNIFORM_COMPONENTS and must be at least 256.

?GL_MAX_VERTEX_OUTPUT_COMPONENTS: Params returns one value, the maximum number of components of output written by a vertex shader, which must be at least 64.

?GL_MAX_VERTEX_UNIFORM_BLOCKS: Params returns one value, the maximum number of uniform blocks per vertex shader. The value must be at least 12. See *gl:uniformBlockBinding/3* .

?GL_MAX_VIEWPORT_DIMS: Params returns two values: the maximum supported width and height of the viewport. These must be at least as large as the visible dimensions of the display being rendered to. See *gl:viewport/4* .

?GL_MAX_VIEWPORTS: Params returns one value, the maximum number of simultaneous viewports that are supported. The value must be at least 16. See *gl:viewportIndexedf/5* .

?GL_MINOR_VERSION: Params returns one value, the minor version number of the OpenGL API supported by the current context.

?GL_NUM_COMPRESSED_TEXTURE_FORMATS: Params returns a single integer value indicating the number of available compressed texture formats. The minimum value is 4. See *gl:compressedTexImage2D/8* .

?GL_NUM_EXTENSIONS: Params returns one value, the number of extensions supported by the GL implementation for the current context. See *gl:getString/1* .

?GL_NUM_PROGRAM_BINARY_FORMATS: Params returns one value, the number of program binary formats supported by the implementation.

?GL_NUM_SHADER_BINARY_FORMATS: Params returns one value, the number of binary shader formats supported by the implementation. If this value is greater than zero, then the implementation supports loading binary shaders. If it is zero, then the loading of binary shaders by the implementation is not supported.

?GL_PACK_ALIGNMENT: Params returns one value, the byte alignment used for writing pixel data to memory. The initial value is 4. See *gl:pixelStoref/2* .

?GL_PACK_IMAGE_HEIGHT: Params returns one value, the image height used for writing pixel data to memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_PACK_LSB_FIRST: Params returns a single boolean value indicating whether single-bit pixels being written to memory are written first to the least significant bit of each unsigned byte. The initial value is ?GL_FALSE. See *gl:pixelStoref/2* .

?GL_PACK_ROW_LENGTH: Params returns one value, the row length used for writing pixel data to memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_PACK_SKIP_IMAGES: Params returns one value, the number of pixel images skipped before the first pixel is written into memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_PACK_SKIP_PIXELS: Params returns one value, the number of pixel locations skipped before the first pixel is written into memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_PACK_SKIP_ROWS: Params returns one value, the number of rows of pixel locations skipped before the first pixel is written into memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_PACK_SWAP_BYTES: Params returns a single boolean value indicating whether the bytes of two-byte and four-byte pixel indices and components are swapped before being written to memory. The initial value is ?GL_FALSE. See *gl:pixelStoref/2* .

?GL_PIXEL_PACK_BUFFER_BINDING: Params returns a single value, the name of the buffer object currently bound to the target ?GL_PIXEL_PACK_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See *gl:bindBuffer/2* .

?GL_PIXEL_UNPACK_BUFFER_BINDING: Params returns a single value, the name of the buffer object currently bound to the target ?GL_PIXEL_UNPACK_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See *gl:bindBuffer/2* .

?GL_POINT_FADE_THRESHOLD_SIZE: Params returns one value, the point size threshold for determining the point size. See *gl:pointParameterf/2* .

?GL_PRIMITIVE_RESTART_INDEX: Params returns one value, the current primitive restart index. The initial value is 0. See *gl:primitiveRestartIndex/1* .

?GL_PROGRAM_BINARY_FORMATS: Params an array of ?GL_NUM_PROGRAM_BINARY_FORMATS values, indicating the proram binary formats supported by the implementation.

?GL_PROGRAM_PIPELINE_BINDING: Params a single value, the name of the currently bound program pipeline object, or zero if no program pipeline object is bound. See *gl:bindProgramPipeline/1* .

?GL_PROVOKING_VERTEX: Params returns one value, the currently selected provoking vertex convention. The initial value is ?GL_LAST_VERTEX_CONVENTION. See *gl:provokingVertex/1* .

?GL_POINT_SIZE: Params returns one value, the point size as specified by *gl:pointSize/1* . The initial value is 1.

?GL_POINT_SIZE_GRANULARITY: Params returns one value, the size difference between adjacent supported sizes for antialiased points. See *gl:pointSize/1* .

?GL_POINT_SIZE_RANGE: Params returns two values: the smallest and largest supported sizes for antialiased points. The smallest size must be at most 1, and the largest size must be at least 1. See *gl:pointSize/1* .

?GL_POLYGON_OFFSET_FACTOR: Params returns one value, the scaling factor used to determine the variable offset that is added to the depth value of each fragment generated when a polygon is rasterized. The initial value is 0. See *gl:polygonOffset/2* .

?GL_POLYGON_OFFSET_UNITS: Params returns one value. This value is multiplied by an implementation-specific value and then added to the depth value of each fragment generated when a polygon is rasterized. The initial value is 0. See *gl:polygonOffset/2* .

?GL_POLYGON_OFFSET_FILL: Params returns a single boolean value indicating whether polygon offset is enabled for polygons in fill mode. The initial value is ?GL_FALSE . See *gl:polygonOffset/2* .

?GL_POLYGON_OFFSET_LINE: Params returns a single boolean value indicating whether polygon offset is enabled for polygons in line mode. The initial value is ?GL_FALSE . See *gl:polygonOffset/2* .

?GL_POLYGON_OFFSET_POINT: Params returns a single boolean value indicating whether polygon offset is enabled for polygons in point mode. The initial value is ?GL_FALSE . See *gl:polygonOffset/2* .

?GL_POLYGON_SMOOTH: Params returns a single boolean value indicating whether antialiasing of polygons is enabled. The initial value is ?GL_FALSE. See *gl:polygonMode/2* .

?GL_POLYGON_SMOOTH_HINT: Params returns one value, a symbolic constant indicating the mode of the polygon antialiasing hint. The initial value is ?GL_DONT_CARE. See *gl:hint/2* .

?GL_READ_BUFFER: Params returns one value, a symbolic constant indicating which color buffer is selected for reading. The initial value is ?GL_BACK if there is a back buffer, otherwise it is ?GL_FRONT. See *gl:readPixels/7* .

?GL_RENDERBUFFER_BINDING: Params returns a single value, the name of the renderbuffer object currently bound to the target ?GL_RENDERBUFFER. If no renderbuffer object is bound to this target, 0 is returned. The initial value is 0. See *gl:bindRenderbuffer/2* .

?GL_SAMPLE_BUFFERS: Params returns a single integer value indicating the number of sample buffers associated with the framebuffer. See *gl:sampleCoverage/2* .

?GL_SAMPLE_COVERAGE_VALUE: Params returns a single positive floating-point value indicating the current sample coverage value. See *gl:sampleCoverage/2* .

?GL_SAMPLE_COVERAGE_INVERT: Params returns a single boolean value indicating if the temporary coverage value should be inverted. See *gl:sampleCoverage/2* .

?GL_SAMPLER_BINDING: Params returns a single value, the name of the sampler object currently bound to the active texture unit. The initial value is 0. See *gl:bindSampler/2* .

?GL_SAMPLES: Params returns a single integer value indicating the coverage mask size. See *gl:sampleCoverage/2* .

?GL_SCISSOR_BOX: Params returns four values: the x and y window coordinates of the scissor box, followed by its width and height. Initially the x and y window coordinates are both 0 and the width and height are set to the size of the window. See *gl:scissor/4* .

?GL_SCISSOR_TEST: Params returns a single boolean value indicating whether scissoring is enabled. The initial value is ?GL_FALSE. See *gl:scissor/4* .

?GL_SHADER_COMPILER: Params returns a single boolean value indicating whether an online shader compiler is present in the implementation. All desktop OpenGL implementations must support online shader compilations, and therefore the value of ?GL_SHADER_COMPILER will always be ?GL_TRUE.

?GL_SMOOTH_LINE_WIDTH_RANGE: Params returns a pair of values indicating the range of widths supported for smooth (antialiased) lines. See *gl:lineWidth/1* .

?GL_SMOOTH_LINE_WIDTH_GRANULARITY: Params returns a single value indicating the level of quantization applied to smooth line width parameters.

?GL_STENCIL_BACK_FAIL: Params returns one value, a symbolic constant indicating what action is taken for back-facing polygons when the stencil test fails. The initial value is ?GL_KEEP. See *gl:stencilOpSeparate/4* .

?GL_STENCIL_BACK_FUNC: Params returns one value, a symbolic constant indicating what function is used for back-facing polygons to compare the stencil reference value with the stencil buffer value. The initial value is ?GL_ALWAYS. See *gl:stencilFuncSeparate/4* .

?GL_STENCIL_BACK_PASS_DEPTH_FAIL: Params returns one value, a symbolic constant indicating what action is taken for back-facing polygons when the stencil test passes, but the depth test fails. The initial value is ?GL_KEEP. See *gl:stencilOpSeparate/4* .

?GL_STENCIL_BACK_PASS_DEPTH_PASS: Params returns one value, a symbolic constant indicating what action is taken for back-facing polygons when the stencil test passes and the depth test passes. The initial value is ?GL_KEEP. See *gl:stencilOpSeparate/4* .

?GL_STENCIL_BACK_REF: Params returns one value, the reference value that is compared with the contents of the stencil buffer for back-facing polygons. The initial value is 0. See *gl:stencilFuncSeparate/4* .

?GL_STENCIL_BACK_VALUE_MASK: Params returns one value, the mask that is used for back-facing polygons to mask both the stencil reference value and the stencil buffer value before they are compared. The initial value is all 1's. See *gl:stencilFuncSeparate/4* .

?GL_STENCIL_BACK_WRITEMASK: Params returns one value, the mask that controls writing of the stencil bitplanes for back-facing polygons. The initial value is all 1's. See *gl:stencilMaskSeparate/2* .

?GL_STENCIL_CLEAR_VALUE: Params returns one value, the index to which the stencil bitplanes are cleared. The initial value is 0. See *gl:clearStencil/1* .

?GL_STENCIL_FAIL: Params returns one value, a symbolic constant indicating what action is taken when the stencil test fails. The initial value is ?GL_KEEP. See *gl:stencilOp/3* . This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See *gl:stencilOpSeparate/4* .

?GL_STENCIL_FUNC: Params returns one value, a symbolic constant indicating what function is used to compare the stencil reference value with the stencil buffer value. The initial value is ?GL_ALWAYS. See *gl:stencilFunc/3* . This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See *gl:stencilFuncSeparate/4* .

?GL_STENCIL_PASS_DEPTH_FAIL: Params returns one value, a symbolic constant indicating what action is taken when the stencil test passes, but the depth test fails. The initial value is ?GL_KEEP. See *gl:stencilOp/3* . This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See *gl:stencilOpSeparate/4* .

?GL_STENCIL_PASS_DEPTH_PASS: Params returns one value, a symbolic constant indicating what action is taken when the stencil test passes and the depth test passes. The initial value is ?GL_KEEP. See *gl:stencilOp/3* . This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See *gl:stencilOpSeparate/4* .

?GL_STENCIL_REF: Params returns one value, the reference value that is compared with the contents of the stencil buffer. The initial value is 0. See *gl:stencilFunc/3* . This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See *gl:stencilFuncSeparate/4* .

?GL_STENCIL_TEST: Params returns a single boolean value indicating whether stencil testing of fragments is enabled. The initial value is ?GL_FALSE. See *gl:stencilFunc/3* and *gl:stencilOp/3* .

?GL_STENCIL_VALUE_MASK: Params returns one value, the mask that is used to mask both the stencil reference value and the stencil buffer value before they are compared. The initial value is all 1's. See *gl:stencilFunc/3* . This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See *gl:stencilFuncSeparate/4* .

?GL_STENCIL_WRITEMASK: Params returns one value, the mask that controls writing of the stencil bitplanes. The initial value is all 1's. See *gl:stencilMask/1* . This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See *gl:stencilMaskSeparate/2* .

?GL_STEREO: Params returns a single boolean value indicating whether stereo buffers (left and right) are supported.

?GL_SUBPIXEL_BITS: Params returns one value, an estimate of the number of bits of subpixel resolution that are used to position rasterized geometry in window coordinates. The value must be at least 4.

?GL_TEXTURE_BINDING_1D: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_1D. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_1D_ARRAY: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_1D_ARRAY. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_2D: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_2D. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_2D_ARRAY: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_2D_ARRAY. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_2D_MULTISAMPLE: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_2D_MULTISAMPLE. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_2D_MULTISAMPLE_ARRAY: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_2D_MULTISAMPLE_ARRAY . The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_3D: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_3D. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_BUFFER: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_BUFFER. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_CUBE_MAP: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_CUBE_MAP. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_BINDING_RECTANGLE: Params returns a single value, the name of the texture currently bound to the target ?GL_TEXTURE_RECTANGLE. The initial value is 0. See *gl:bindTexture/2* .

?GL_TEXTURE_COMPRESSION_HINT: Params returns a single value indicating the mode of the texture compression hint. The initial value is ?GL_DONT_CARE.

?GL_TEXTURE_BUFFER_BINDING: Params returns a single value, the name of the texture buffer object currently bound. The initial value is 0. See *gl:bindBuffer/2* .

?GL_TIMESTAMP: Params returns a single value, the 64-bit value of the current GL time. See *gl:queryCounter/2* .

?GL_TRANSFORM_FEEDBACK_BUFFER_BINDING: When used with non-indexed variants of gl:get (such as gl:getIntegerv), Params returns a single value, the name of the buffer object currently bound to the target ?GL_TRANSFORM_FEEDBACK_BUFFER. If no buffer object is bound to this target, 0 is returned. When used with indexed variants of gl:get (such as gl:getIntegeri_v), Params returns a single value, the name of the buffer object bound to the indexed transform feedback attribute stream. The initial value is 0 for all targets. See *gl:bindBuffer/2* , *gl:bindBufferBase/3* , and *gl:bindBufferRange/5* .

?GL_TRANSFORM_FEEDBACK_BUFFER_START: When used with indexed variants of gl:get (such as gl:getInteger64i_v), Params returns a single value, the start offset of the binding range for each transform feedback attribute stream. The initial value is 0 for all streams. See *gl:bindBufferRange/5* .

?GL_TRANSFORM_FEEDBACK_BUFFER_SIZE: When used with indexed variants of gl:get (such as gl:getInteger64i_v), Params returns a single value, the size of the binding range for each transform feedback attribute stream. The initial value is 0 for all streams. See *gl:bindBufferRange/5* .

?GL_UNIFORM_BUFFER_BINDING: When used with non-indexed variants of gl:get (such as gl:getIntegerv), Params returns a single value, the name of the buffer object currently bound to the target ?GL_UNIFORM_BUFFER. If no buffer object is bound to this target, 0 is returned. When used with indexed variants of gl:get (such as gl:getIntegeri_v), Params returns a single value, the name of the buffer object bound to the indexed uniform buffer binding point. The initial value is 0 for all targets. See *gl:bindBuffer/2* , *gl:bindBufferBase/3* , and *gl:bindBufferRange/5* .

?GL_UNIFORM_BUFFER_OFFSET_ALIGNMENT: Params returns a single value, the minimum required alignment for uniform buffer sizes and offset. The initial value is 1. See *gl:uniformBlockBinding/3* .

?GL_UNIFORM_BUFFER_SIZE: When used with indexed variants of gl:get (such as gl:getInteger64i_v), Params returns a single value, the size of the binding range for each indexed uniform buffer binding. The initial value is 0 for all bindings. See *gl:bindBufferRange/5* .

?GL_UNIFORM_BUFFER_START: When used with indexed variants of gl:get (such as gl:getInteger64i_v), Params returns a single value, the start offset of the binding range for each indexed uniform buffer binding. The initial value is 0 for all bindings. See *gl:bindBufferRange/5* .

?GL_UNPACK_ALIGNMENT: Params returns one value, the byte alignment used for reading pixel data from memory. The initial value is 4. See *gl:pixelStoref/2* .

?GL_UNPACK_IMAGE_HEIGHT: Params returns one value, the image height used for reading pixel data from memory. The initial is 0. See *gl:pixelStoref/2* .

?GL_UNPACK_LSB_FIRST: Params returns a single boolean value indicating whether single-bit pixels being read from memory are read first from the least significant bit of each unsigned byte. The initial value is ?GL_FALSE. See *gl:pixelStoref/2* .

?GL_UNPACK_ROW_LENGTH: Params returns one value, the row length used for reading pixel data from memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_UNPACK_SKIP_IMAGES: Params returns one value, the number of pixel images skipped before the first pixel is read from memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_UNPACK_SKIP_PIXELS: Params returns one value, the number of pixel locations skipped before the first pixel is read from memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_UNPACK_SKIP_ROWS: Params returns one value, the number of rows of pixel locations skipped before the first pixel is read from memory. The initial value is 0. See *gl:pixelStoref/2* .

?GL_UNPACK_SWAP_BYTES: Params returns a single boolean value indicating whether the bytes of two-byte and four-byte pixel indices and components are swapped after being read from memory. The initial value is ?GL_FALSE. See *gl:pixelStoref/2* .

?GL_VERTEX_PROGRAM_POINT_SIZE: Params returns a single boolean value indicating whether vertex program point size mode is enabled. If enabled, and a vertex shader is active, then the point size is taken from the shader built-in gl_PointSize. If disabled, and a vertex shader is active, then the point size is taken from the point state as specified by *gl:pointSize/1* . The initial value is ?GL_FALSE.

?GL_VIEWPORT: When used with non-indexed variants of gl:get (such as gl:getIntegerv), Params returns four values: the x and y window coordinates of the viewport, followed by its width and height. Initially the x and y window coordinates are both set to 0, and the width and height are set to the width and height of the window into which the GL will do its rendering. See *gl:viewport/4* . When used with indexed variants of gl:get (such as gl:getIntegeri_v), Params returns four values: the x and y window coordinates of the indexed viewport, followed by its width and height. Initially the x and y window coordinates are both set to 0, and the width and height are set to the width and height of the window into which the GL will do its rendering. See *gl:viewportIndexedf/5* .

?GL_VIEWPORT_BOUNDS_RANGE: Params returns two values, the minimum and maximum viewport bounds range. The minimum range should be at least [-32768, 32767].

?GL_VIEWPORT_INDEX_PROVOKING_VERTEX: Params returns one value, the implementation dependent specifc vertex of a primitive that is used to select the viewport index. If the value returned is equivalent to ?GL_PROVOKING_VERTEX, then the vertex selection follows the convention specified by *gl:provokingVertex/1* . If the value returned is equivalent to ?GL_FIRST_VERTEX_CONVENTION, then the selection is always taken from the first vertex in the primitive. If the value returned is equivalent to ?GL_LAST_VERTEX_CONVENTION , then the selection is always taken from the last vertex in the primitive. If the value returned is equivalent to ?GL_UNDEFINED_VERTEX, then the selection is not guaranteed to be taken from any specific vertex in the primitive.

?GL_VIEWPORT_SUBPIXEL_BITS: Params returns a single value, the number of bits of sub-pixel precision which the GL uses to interpret the floating point viewport bounds. The minimum value is 0.

Many of the boolean parameters can also be queried more easily using *gl:isEnabled/1* .

See **external** documentation.

getDoublev(Pname) -> [float()]

Types:

Pname = enum()

See *getBooleanv/1*

getFloatv(Pname) -> [float()]

Types:

Pname = enum()

See *getBooleanv/1*

getIntegerv(Pname) -> [integer()]

Types:

Pname = enum()

See *getBooleanv/1*

pushAttrib(Mask) -> ok

Types:

Mask = integer()

Push and pop the server attribute stack

`gl:pushAttrib` takes one argument, a mask that indicates which groups of state variables to save on the attribute stack. Symbolic constants are used to set bits in the mask. `Mask` is typically constructed by specifying the bitwise-or of several of these constants together. The special mask `?GL_ALL_ATTRIB_BITS` can be used to save all stackable states.

The symbolic mask constants and their associated GL state are as follows (the second column lists which attributes are saved):

`?GL_ACCUM_BUFFER_BIT` Accumulation buffer clear value
`?GL_COLOR_BUFFER_BIT?GL_ALPHA_TEST` enable bit
Alpha test function and reference value
`?GL_BLEND` enable bit
Blending source and destination functions
Constant blend color
Blending equation
`?GL_DITHER` enable bit
`?GL_DRAW_BUFFER` setting
`?GL_COLOR_LOGIC_OP` enable bit
`?GL_INDEX_LOGIC_OP` enable bit
Logic op function
Color mode and index mode clear values
Color mode and index mode writemasks
`?GL_CURRENT_BIT` Current RGBA color
Current color index
Current normal vector
Current texture coordinates
Current raster position
`?GL_CURRENT_RASTER_POSITION_VALID` flag
RGBA color associated with current raster position
Color index associated with current raster position
Texture coordinates associated with current raster position
`?GL_EDGE_FLAG` flag
`?GL_DEPTH_BUFFER_BIT?GL_DEPTH_TEST` enable bit
Depth buffer test function
Depth buffer clear value
`?GL_DEPTH_WRITEMASK` enable bit
`?GL_ENABLE_BIT?GL_ALPHA_TEST` flag
`?GL_AUTO_NORMAL` flag
`?GL_BLEND` flag
Enable bits for the user-definable clipping planes
`?GL_COLOR_MATERIAL`
`?GL_CULL_FACE` flag
`?GL_DEPTH_TEST` flag
`?GL_DITHER` flag
`?GL_FOG` flag
`?GL_LIGHT i` where `?0` <= `i` < `?GL_MAX_LIGHTS`
`?GL_LIGHTING` flag
`?GL_LINE_SMOOTH` flag

?GL_LINE_STIPPLE flag
?GL_COLOR_LOGIC_OP flag
?GL_INDEX_LOGIC_OP flag
?GL_MAP1_x where x is a map type
?GL_MAP2_ x where x is a map type
?GL_MULTISAMPLE flag
?GL_NORMALIZE flag
?GL_POINT_SMOOTH flag
?GL_POLYGON_OFFSET_LINE flag
?GL_POLYGON_OFFSET_FILL flag
?GL_POLYGON_OFFSET_POINT flag
?GL_POLYGON_SMOOTH flag
?GL_POLYGON_STIPPLE flag
?GL_SAMPLE_ALPHA_TO_COVERAGE flag
?GL_SAMPLE_ALPHA_TO_ONE flag
?GL_SAMPLE_COVERAGE flag
?GL_SCISSOR_TEST flag
?GL_STENCIL_TEST flag
?GL_TEXTURE_1D flag
?GL_TEXTURE_2D flag
?GL_TEXTURE_3D flag
Flags ?GL_TEXTURE_GEN_x where x is S, T, R, or Q
?GL_EVAL_BIT?GL_MAP1_x enable bits, where x is a map type
?GL_MAP2_x enable bits, where x is a map type
1D grid endpoints and divisions
2D grid endpoints and divisions
?GL_AUTO_NORMAL enable bit
?GL_FOG_BIT?GL_FOG enable bit
Fog color
Fog density
Linear fog start
Linear fog end
Fog index
?GL_FOG_MODE value
?GL_HINT_BIT?GL_PERSPECTIVE_CORRECTION_HINT setting
?GL_POINT_SMOOTH_HINT setting
?GL_LINE_SMOOTH_HINT setting
?GL_POLYGON_SMOOTH_HINT setting
?GL_FOG_HINT setting
?GL_GENERATE_MIPMAP_HINT setting
?GL_TEXTURE_COMPRESSION_HINT setting
?GL_LIGHTING_BIT?GL_COLOR_MATERIAL enable bit
?GL_COLOR_MATERIAL_FACE value
Color material parameters that are tracking the current color
Ambient scene color
?GL_LIGHT_MODEL_LOCAL_VIEWER value
?GL_LIGHT_MODEL_TWO_SIDE setting
?GL_LIGHTING enable bit
Enable bit for each light
Ambient, diffuse, and specular intensity for each light
Direction, position, exponent, and cutoff angle for each light
Constant, linear, and quadratic attenuation factors for each light

Ambient, diffuse, specular, and emissive color for each material
Ambient, diffuse, and specular color indices for each material
Specular exponent for each material
?GL_SHADE_MODEL setting
?GL_LINE_BIT?GL_LINE_SMOOTH flag
?GL_LINE_STIPPLE enable bit
Line stipple pattern and repeat counter
Line width
?GL_LIST_BIT ?GL_LIST_BASE setting
?GL_MULTISAMPLE_BIT ?GL_MULTISAMPLE flag
?GL_SAMPLE_ALPHA_TO_COVERAGE flag
?GL_SAMPLE_ALPHA_TO_ONE flag
?GL_SAMPLE_COVERAGE flag
?GL_SAMPLE_COVERAGE_VALUE value
?GL_SAMPLE_COVERAGE_INVERT value
?GL_PIXEL_MODE_BIT?GL_RED_BIAS and ?GL_RED_SCALE settings
?GL_GREEN_BIAS and ?GL_GREEN_SCALE values
?GL_BLUE_BIAS and ?GL_BLUE_SCALE
?GL_ALPHA_BIAS and ?GL_ALPHA_SCALE
?GL_DEPTH_BIAS and ?GL_DEPTH_SCALE
?GL_INDEX_OFFSET and ?GL_INDEX_SHIFT values
?GL_MAP_COLOR and ?GL_MAP_STENCIL flags
?GL_ZOOM_X and ?GL_ZOOM_Y factors
?GL_READ_BUFFER setting
?GL_POINT_BIT?GL_POINT_SMOOTH flag
Point size
?GL_POLYGON_BIT?GL_CULL_FACE enable bit
?GL_CULL_FACE_MODE value
?GL_FRONT_FACE indicator
?GL_POLYGON_MODE setting
?GL_POLYGON_SMOOTH flag
?GL_POLYGON_STIPPLE enable bit
?GL_POLYGON_OFFSET_FILL flag
?GL_POLYGON_OFFSET_LINE flag
?GL_POLYGON_OFFSET_POINT flag
?GL_POLYGON_OFFSET_FACTOR
?GL_POLYGON_OFFSET_UNITS
?GL_POLYGON_STIPPLE_BIT Polygon stipple image
?GL_SCISSOR_BIT?GL_SCISSOR_TEST flag
Scissor box
?GL_STENCIL_BUFFER_BIT ?GL_STENCIL_TEST enable bit
Stencil function and reference value
Stencil value mask
Stencil fail, pass, and depth buffer pass actions
Stencil buffer clear value
Stencil buffer writemask
?GL_TEXTURE_BIT Enable bits for the four texture coordinates
Border color for each texture image
Minification function for each texture image
Magnification function for each texture image
Texture coordinates and wrap mode for each texture image
Color and mode for each texture environment

Enable bits ?GL_TEXTURE_GEN_x, x is S, T, R, and Q
?GL_TEXTURE_GEN_MODE setting for S, T, R, and Q
gl:texGend/3 plane equations for S, T, R, and Q
Current texture bindings (for example, ?GL_TEXTURE_BINDING_2D)
?GL_TRANSFORM_BIT Coefficients of the six clipping planes
Enable bits for the user-definable clipping planes
?GL_MATRIX_MODE value
?GL_NORMALIZE flag
?GL_RESCALE_NORMAL flag
?GL_VIEWPORT_BIT Depth range (near and far)
Viewport origin and extent

gl:pushAttrib/1 restores the values of the state variables saved with the last gl:pushAttrib command. Those not saved are left unchanged.

It is an error to push attributes onto a full stack or to pop attributes off an empty stack. In either case, the error flag is set and no other change is made to GL state.

Initially, the attribute stack is empty.

See **external** documentation.

popAttrib() -> ok

See *pushAttrib/1*

pushClientAttrib(Mask) -> ok

Types:

```
Mask = integer()
```

Push and pop the client attribute stack

gl:pushClientAttrib takes one argument, a mask that indicates which groups of client-state variables to save on the client attribute stack. Symbolic constants are used to set bits in the mask. Mask is typically constructed by specifying the bitwise-or of several of these constants together. The special mask ?GL_CLIENT_ALL_ATTRIB_BITS can be used to save all stackable client state.

The symbolic mask constants and their associated GL client state are as follows (the second column lists which attributes are saved):

?GL_CLIENT_PIXEL_STORE_BIT Pixel storage modes ?GL_CLIENT_VERTEX_ARRAY_BIT Vertex arrays (and enables)

gl:pushClientAttrib/1 restores the values of the client-state variables saved with the last gl:pushClientAttrib. Those not saved are left unchanged.

It is an error to push attributes onto a full client attribute stack or to pop attributes off an empty stack. In either case, the error flag is set, and no other change is made to GL state.

Initially, the client attribute stack is empty.

See **external** documentation.

popClientAttrib() -> ok

See *pushClientAttrib/1*

renderMode(Mode) -> integer()

Types:

Mode = enum()

Set rasterization mode

`gl:renderMode` sets the rasterization mode. It takes one argument, `Mode` , which can assume one of three predefined values:

`?GL_RENDER`: Render mode. Primitives are rasterized, producing pixel fragments, which are written into the frame buffer. This is the normal mode and also the default mode.

`?GL_SELECT`: Selection mode. No pixel fragments are produced, and no change to the frame buffer contents is made. Instead, a record of the names of primitives that would have been drawn if the render mode had been `?GL_RENDER` is returned in a select buffer, which must be created (see *gl:selectBuffer/2*) before selection mode is entered.

`?GL_FEEDBACK`: Feedback mode. No pixel fragments are produced, and no change to the frame buffer contents is made. Instead, the coordinates and attributes of vertices that would have been drawn if the render mode had been `?GL_RENDER` is returned in a feedback buffer, which must be created (see *gl:feedbackBuffer/3*) before feedback mode is entered.

The return value of `gl:renderMode` is determined by the render mode at the time `gl:renderMode` is called, rather than by `Mode` . The values returned for the three render modes are as follows:

`?GL_RENDER`: 0.

`?GL_SELECT`: The number of hit records transferred to the select buffer.

`?GL_FEEDBACK`: The number of values (not vertices) transferred to the feedback buffer.

See the *gl:selectBuffer/2* and *gl:feedbackBuffer/3* reference pages for more details concerning selection and feedback operation.

See **external** documentation.

getError() -> enum()

Return error information

`gl:getError` returns the value of the error flag. Each detectable error is assigned a numeric code and symbolic name. When an error occurs, the error flag is set to the appropriate error code value. No other errors are recorded until `gl:getError` is called, the error code is returned, and the flag is reset to `?GL_NO_ERROR`. If a call to `gl:getError` returns `?GL_NO_ERROR`, there has been no detectable error since the last call to `gl:getError` , or since the GL was initialized.

To allow for distributed implementations, there may be several error flags. If any single error flag has recorded an error, the value of that flag is returned and that flag is reset to `?GL_NO_ERROR` when `gl:getError` is called. If more than one flag has recorded an error, `gl:getError` returns and clears an arbitrary error flag value. Thus, `gl:getError` should always be called in a loop, until it returns `?GL_NO_ERROR` , if all error flags are to be reset.

Initially, all error flags are set to `?GL_NO_ERROR`.

The following errors are currently defined:

`?GL_NO_ERROR`: No error has been recorded. The value of this symbolic constant is guaranteed to be 0.

`?GL_INVALID_ENUM`: An unacceptable value is specified for an enumerated argument. The offending command is ignored and has no other side effect than to set the error flag.

`?GL_INVALID_VALUE`: A numeric argument is out of range. The offending command is ignored and has no other side effect than to set the error flag.

?GL_INVALID_OPERATION: The specified operation is not allowed in the current state. The offending command is ignored and has no other side effect than to set the error flag.

?GL_INVALID_FRAMEBUFFER_OPERATION: The framebuffer object is not complete. The offending command is ignored and has no other side effect than to set the error flag.

?GL_OUT_OF_MEMORY: There is not enough memory left to execute the command. The state of the GL is undefined, except for the state of the error flags, after this error is recorded.

When an error flag is set, results of a GL operation are undefined only if ?GL_OUT_OF_MEMORY has occurred. In all other cases, the command generating the error is ignored and has no effect on the GL state or frame buffer contents. If the generating command returns a value, it returns 0. If gl:getError itself generates an error, it returns 0.

See **external** documentation.

getString(Name) -> string()

Types:

Name = enum()

Return a string describing the current GL connection

gl:getString returns a pointer to a static string describing some aspect of the current GL connection. Name can be one of the following:

?GL_VENDOR: Returns the company responsible for this GL implementation. This name does not change from release to release.

?GL_RENDERER: Returns the name of the renderer. This name is typically specific to a particular configuration of a hardware platform. It does not change from release to release.

?GL_VERSION: Returns a version or release number.

?GL_SHADING_LANGUAGE_VERSION: Returns a version or release number for the shading language.

gl:getStringi returns a pointer to a static string indexed by Index . Name can be one of the following:

?GL_EXTENSIONS: For gl:getStringi only, returns the extension string supported by the implementation at Index .

Strings ?GL_VENDOR and ?GL_RENDERER together uniquely specify a platform. They do not change from release to release and should be used by platform-recognition algorithms.

The ?GL_VERSION and ?GL_SHADING_LANGUAGE_VERSION strings begin with a version number. The version number uses one of these forms:

major_number.minor_numbermajor_number.minor_number.release_number

Vendor-specific information may follow the version number. Its format depends on the implementation, but a space always separates the version number and the vendor-specific information.

All strings are null-terminated.

See **external** documentation.

finish() -> ok

Block until all GL execution is complete

gl:finish does not return until the effects of all previously called GL commands are complete. Such effects include all changes to GL state, all changes to connection state, and all changes to the frame buffer contents.

See **external** documentation.

flush() -> ok

Force execution of GL commands in finite time

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. `gl:flush` empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

Because any GL program might be executed over a network, or on an accelerator that buffers commands, all programs should call `gl:flush` whenever they count on having all of their previously issued commands completed. For example, call `gl:flush` before waiting for user input that depends on the generated image.

See **external** documentation.

hint(Target, Mode) -> ok

Types:

```
Target = enum()
Mode = enum()
```

Specify implementation-specific hints

Certain aspects of GL behavior, when there is room for interpretation, can be controlled with hints. A hint is specified with two arguments. `Target` is a symbolic constant indicating the behavior to be controlled, and `Mode` is another symbolic constant indicating the desired behavior. The initial value for each `Target` is `?GL_DONT_CARE` . `Mode` can be one of the following:

`?GL_FASTEST`: The most efficient option should be chosen.

`?GL_NICEST`: The most correct, or highest quality, option should be chosen.

`?GL_DONT_CARE`: No preference.

Though the implementation aspects that can be hinted are well defined, the interpretation of the hints depends on the implementation. The hint aspects that can be specified with `Target` , along with suggested semantics, are as follows:

`?GL_FRAGMENT_SHADER_DERIVATIVE_HINT`: Indicates the accuracy of the derivative calculation for the GL shading language fragment processing built-in functions: `?dFdx` , `?dFdy`, and `?fwidth`.

`?GL_LINE_SMOOTH_HINT`: Indicates the sampling quality of antialiased lines. If a larger filter function is applied, hinting `?GL_NICEST` can result in more pixel fragments being generated during rasterization.

`?GL_POLYGON_SMOOTH_HINT`: Indicates the sampling quality of antialiased polygons. Hinting `?GL_NICEST` can result in more pixel fragments being generated during rasterization, if a larger filter function is applied.

`?GL_TEXTURE_COMPRESSION_HINT`: Indicates the quality and performance of the compressing texture images. Hinting `?GL_FASTEST` indicates that texture images should be compressed as quickly as possible, while `?GL_NICEST` indicates that texture images should be compressed with as little image quality loss as possible. `?GL_NICEST` should be selected if the texture is to be retrieved by *gl:getCompressedTexImage/3* for reuse.

See **external** documentation.

clearDepth(Depth) -> ok

Types:

```
Depth = clamp()
```

Specify the clear value for the depth buffer

`gl:clearDepth` specifies the depth value used by *gl:clear/1* to clear the depth buffer. Values specified by `gl:clearDepth` are clamped to the range [0 1].

See **external** documentation.

depthFunc(Func) -> ok

Types:

Func = enum()

Specify the value used for depth buffer comparisons

`gl:depthFunc` specifies the function used to compare each incoming pixel depth value with the depth value present in the depth buffer. The comparison is performed only if depth testing is enabled. (See *gl:enable/1* and *gl:enable/1* of `?GL_DEPTH_TEST` .)

`Func` specifies the conditions under which the pixel will be drawn. The comparison functions are as follows:

`?GL_NEVER`: Never passes.

`?GL_LESS`: Passes if the incoming depth value is less than the stored depth value.

`?GL_EQUAL`: Passes if the incoming depth value is equal to the stored depth value.

`?GL_LEQUAL`: Passes if the incoming depth value is less than or equal to the stored depth value.

`?GL_GREATER`: Passes if the incoming depth value is greater than the stored depth value.

`?GL_NOTEQUAL`: Passes if the incoming depth value is not equal to the stored depth value.

`?GL_GEQUAL`: Passes if the incoming depth value is greater than or equal to the stored depth value.

`?GL_ALWAYS`: Always passes.

The initial value of `Func` is `?GL_LESS`. Initially, depth testing is disabled. If depth testing is disabled or if no depth buffer exists, it is as if the depth test always passes.

See **external** documentation.

depthMask(Flag) -> ok

Types:

Flag = 0 | 1

Enable or disable writing into the depth buffer

`gl:depthMask` specifies whether the depth buffer is enabled for writing. If `Flag` is `?GL_FALSE`, depth buffer writing is disabled. Otherwise, it is enabled. Initially, depth buffer writing is enabled.

See **external** documentation.

depthRange(Near_val, Far_val) -> ok

Types:

Near_val = clamp()

Far_val = clamp()

Specify mapping of depth values from normalized device coordinates to window coordinates

After clipping and division by `w`, depth coordinates range from -1 to 1, corresponding to the near and far clipping planes. `gl:depthRange` specifies a linear mapping of the normalized depth coordinates in this range to window depth coordinates. Regardless of the actual depth buffer implementation, window coordinate depth values are treated as though they range from 0 through 1 (like color components). Thus, the values accepted by `gl:depthRange` are both clamped to this range before they are accepted.

The setting of (0,1) maps the near plane to 0 and the far plane to 1. With this mapping, the depth buffer range is fully utilized.

See **external** documentation.

clearAccum(Red, Green, Blue, Alpha) -> ok

Types:

Red = float()
Green = float()
Blue = float()
Alpha = float()

Specify clear values for the accumulation buffer

`gl:clearAccum` specifies the red, green, blue, and alpha values used by *gl:clear/1* to clear the accumulation buffer.

Values specified by `gl:clearAccum` are clamped to the range [-1 1].

See **external** documentation.

accum(Op, Value) -> ok

Types:

Op = enum()
Value = float()

Operate on the accumulation buffer

The accumulation buffer is an extended-range color buffer. Images are not rendered into it. Rather, images rendered into one of the color buffers are added to the contents of the accumulation buffer after rendering. Effects such as antialiasing (of points, lines, and polygons), motion blur, and depth of field can be created by accumulating images generated with different transformation matrices.

Each pixel in the accumulation buffer consists of red, green, blue, and alpha values. The number of bits per component in the accumulation buffer depends on the implementation. You can examine this number by calling *gl:getBooleanv/1* four times, with arguments `?GL_ACCUM_RED_BITS`, `?GL_ACCUM_GREEN_BITS`, `?GL_ACCUM_BLUE_BITS`, and `?GL_ACCUM_ALPHA_BITS` . Regardless of the number of bits per component, the range of values stored by each component is [-1 1]. The accumulation buffer pixels are mapped one-to-one with frame buffer pixels.

`gl:accum` operates on the accumulation buffer. The first argument, `Op` , is a symbolic constant that selects an accumulation buffer operation. The second argument, `Value` , is a floating-point value to be used in that operation. Five operations are specified: `?GL_ACCUM` , `?GL_LOAD`, `?GL_ADD`, `?GL_MULT`, and `?GL_RETURN`.

All accumulation buffer operations are limited to the area of the current scissor box and applied identically to the red, green, blue, and alpha components of each pixel. If a `gl:accum` operation results in a value outside the range [-1 1], the contents of an accumulation buffer pixel component are undefined.

The operations are as follows:

`?GL_ACCUM`: Obtains R, G, B, and A values from the buffer currently selected for reading (see *gl:readBuffer/1*). Each component value is divided by 2 n-1, where n is the number of bits allocated to each color component in the currently selected buffer. The result is a floating-point value in the range [0 1], which is multiplied by `Value` and added to the corresponding pixel component in the accumulation buffer, thereby updating the accumulation buffer.

`?GL_LOAD`: Similar to `?GL_ACCUM`, except that the current value in the accumulation buffer is not used in the calculation of the new value. That is, the R, G, B, and A values from the currently selected buffer are divided by 2 n-1, multiplied by `Value` , and then stored in the corresponding accumulation buffer cell, overwriting the current value.

`?GL_ADD`: Adds `Value` to each R, G, B, and A in the accumulation buffer.

`?GL_MULT`: Multiplies each R, G, B, and A in the accumulation buffer by `Value` and returns the scaled component to its corresponding accumulation buffer location.

?GL_RETURN: Transfers accumulation buffer values to the color buffer or buffers currently selected for writing. Each R, G, B, and A component is multiplied by Value , then multiplied by 2 n-1, clamped to the range [0 2 n-1], and stored in the corresponding display buffer cell. The only fragment operations that are applied to this transfer are pixel ownership, scissor, dithering, and color writemasks.

To clear the accumulation buffer, call *gl:clearAccum/4* with R, G, B, and A values to set it to, then call *gl:clear/1* with the accumulation buffer enabled.

See **external** documentation.

matrixMode(Mode) -> ok

Types:

```
Mode = enum()
```

Specify which matrix is the current matrix

gl:matrixMode sets the current matrix mode. Mode can assume one of four values:

?GL_MODELVIEW: Applies subsequent matrix operations to the modelview matrix stack.

?GL_PROJECTION: Applies subsequent matrix operations to the projection matrix stack.

?GL_TEXTURE: Applies subsequent matrix operations to the texture matrix stack.

?GL_COLOR: Applies subsequent matrix operations to the color matrix stack.

To find out which matrix stack is currently the target of all matrix operations, call *gl:getBooleanv/1* with argument ?GL_MATRIX_MODE. The initial value is ?GL_MODELVIEW.

See **external** documentation.

ortho(Left, Right, Bottom, Top, Near_val, Far_val) -> ok

Types:

```
Left = float()
Right = float()
Bottom = float()
Top = float()
Near_val = float()
Far_val = float()
```

Multiply the current matrix with an orthographic matrix

gl:ortho describes a transformation that produces a parallel projection. The current matrix (see *gl:matrixMode/1*) is multiplied by this matrix and the result replaces the current matrix, as if *gl:multMatrixd/1* were called with the following matrix as its argument:

((2/(right-left)) 0 0(t x) 0(2/(top-bottom)) 0(t y) 0 0(-2/(farVal-nearVal))(t z) 0 0 0 1)

where t x=-((right+left)/(right-left)) t y=-((top+bottom)/(top-bottom)) t z=-((farVal+nearVal)/(farVal-nearVal))

Typically, the matrix mode is ?GL_PROJECTION, and (left bottom-nearVal) and (right top-nearVal) specify the points on the near clipping plane that are mapped to the lower left and upper right corners of the window, respectively, assuming that the eye is located at (0, 0, 0). -farVal specifies the location of the far clipping plane. Both NearVal and FarVal can be either positive or negative.

Use *gl:pushMatrix/0* and *gl:pushMatrix/0* to save and restore the current matrix stack.

See **external** documentation.

frustum(Left, Right, Bottom, Top, Near_val, Far_val) -> ok

Types:

Left = float()
Right = float()
Bottom = float()
Top = float()
Near_val = float()
Far_val = float()

Multiply the current matrix by a perspective matrix

`gl:frustum` describes a perspective matrix that produces a perspective projection. The current matrix (see *gl:matrixMode/1*) is multiplied by this matrix and the result replaces the current matrix, as if *gl:multMatrixd/1* were called with the following matrix as its argument:

[((2 nearVal)/(right-left)) 0 A 0 0((2 nearVal)/(top-bottom)) B 0 0 0 C D 0 0 -1 0]

A=(right+left)/(right-left)

B=(top+bottom)/(top-bottom)

C=-((farVal+nearVal)/(farVal-nearVal))

D=-((2 farVal nearVal)/(farVal-nearVal))

Typically, the matrix mode is `?GL_PROJECTION`, and (left bottom-nearVal) and (right top-nearVal) specify the points on the near clipping plane that are mapped to the lower left and upper right corners of the window, assuming that the eye is located at (0, 0, 0). -farVal specifies the location of the far clipping plane. Both `NearVal` and `FarVal` must be positive.

Use *gl:pushMatrix/0* and *gl:pushMatrix/0* to save and restore the current matrix stack.

See **external** documentation.

viewport(X, Y, Width, Height) -> ok

Types:

X = integer()
Y = integer()
Width = integer()
Height = integer()

Set the viewport

`gl:viewport` specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let (x nd y nd) be normalized device coordinates. Then the window coordinates (x w y w) are computed as follows:

x w=(x nd+1) (width/2)+x

y w=(y nd+1) (height/2)+y

Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, call *gl:getBooleanv/1* with argument `?GL_MAX_VIEWPORT_DIMS`.

See **external** documentation.

pushMatrix() -> ok

Push and pop the current matrix stack

There is a stack of matrices for each of the matrix modes. In `?GL_MODELVIEW` mode, the stack depth is at least 32. In the other modes, `?GL_COLOR`, `?GL_PROJECTION` , and `?GL_TEXTURE`, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

`gl:pushMatrix` pushes the current matrix stack down by one, duplicating the current matrix. That is, after a `gl:pushMatrix` call, the matrix on top of the stack is identical to the one below it.

gl:pushMatrix/0 pops the current matrix stack, replacing the current matrix with the one below it on the stack.

Initially, each of the stacks contains one matrix, an identity matrix.

It is an error to push a full matrix stack or to pop a matrix stack that contains only a single matrix. In either case, the error flag is set and no other change is made to GL state.

See **external** documentation.

popMatrix() -> ok

See *pushMatrix/0*

loadIdentity() -> ok

Replace the current matrix with the identity matrix

`gl:loadIdentity` replaces the current matrix with the identity matrix. It is semantically equivalent to calling *gl:loadMatrixd/1* with the identity matrix

((1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1))

but in some cases it is more efficient.

See **external** documentation.

loadMatrixd(M) -> ok

Types:

```
M = matrix()
```

Replace the current matrix with the specified matrix

`gl:loadMatrix` replaces the current matrix with the one whose elements are specified by `M` . The current matrix is the projection matrix, modelview matrix, or texture matrix, depending on the current matrix mode (see *gl:matrixMode/1*).

The current matrix, M, defines a transformation of coordinates. For instance, assume M refers to the modelview matrix. If v=(v[0] v[1] v[2] v[3]) is the set of object coordinates of a vertex, and `M` points to an array of 16 single- or double-precision floating-point values m={m[0] m[1] ... m[15]}, then the modelview transformation M(v) does the following:

M(v)=(m[0] m[4] m[8] m[12] m[1] m[5] m[9] m[13] m[2] m[6] m[10] m[14] m[3] m[7] m[11] m[15])*(v[0] v[1] v[2] v[3])

Projection and texture transformations are similarly defined.

See **external** documentation.

loadMatrixf(M) -> ok

Types:

```
M = matrix()
```

See *loadMatrixd/1*

multMatrixd(M) -> ok

Types:

M = matrix()

Multiply the current matrix with the specified matrix

`gl:multMatrix` multiplies the current matrix with the one specified using `M` , and replaces the current matrix with the product.

The current matrix is determined by the current matrix mode (see *gl:matrixMode/1*). It is either the projection matrix, modelview matrix, or the texture matrix.

See **external** documentation.

multMatrixf(M) -> ok

Types:

M = matrix()

See *multMatrixd/1*

rotated(Angle, X, Y, Z) -> ok

Types:

Angle = float()
X = float()
Y = float()
Z = float()

Multiply the current matrix by a rotation matrix

`gl:rotate` produces a rotation of `Angle` degrees around the vector (x y z). The current matrix (see *gl:matrixMode/1*) is multiplied by a rotation matrix with the product replacing the current matrix, as if *gl:multMatrixd/1* were called with the following matrix as its argument:

(x 2(1-c)+c x y(1-c)-z s x z(1-c)+y s 0 y x(1-c)+z s y 2(1-c)+c y z(1-c)-x s 0 x z(1-c)-y s y z(1-c)+x s z 2(1-c)+c 0 0 0 0 1)

Where c= cos(angle), s= sin(angle), and ||(x y z)||= 1 (if not, the GL will normalize this vector).

If the matrix mode is either `?GL_MODELVIEW` or `?GL_PROJECTION`, all objects drawn after `gl:rotate` is called are rotated. Use *gl:pushMatrix/0* and *gl:pushMatrix/0* to save and restore the unrotated coordinate system.

See **external** documentation.

rotatef(Angle, X, Y, Z) -> ok

Types:

Angle = float()
X = float()
Y = float()
Z = float()

See *rotated/4*

scaled(X, Y, Z) -> ok

Types:

X = float()

Y = float()
Z = float()

Multiply the current matrix by a general scaling matrix

`gl:scale` produces a nonuniform scaling along the `x`, `y`, and `z` axes. The three parameters indicate the desired scale factor along each of the three axes.

The current matrix (see *gl:matrixMode/1*) is multiplied by this scale matrix, and the product replaces the current matrix as if *gl:multMatrixd/1* were called with the following matrix as its argument:

(x 0 0 0 0 y 0 0 0 0 z 0 0 0 0 1)

If the matrix mode is either `?GL_MODELVIEW` or `?GL_PROJECTION`, all objects drawn after `gl:scale` is called are scaled.

Use *gl:pushMatrix/0* and *gl:pushMatrix/0* to save and restore the unscaled coordinate system.

See **external** documentation.

scalef(X, Y, Z) -> ok

Types:

X = float()
Y = float()
Z = float()

See *scaled/3*

translated(X, Y, Z) -> ok

Types:

X = float()
Y = float()
Z = float()

Multiply the current matrix by a translation matrix

`gl:translate` produces a translation by (x y z). The current matrix (see *gl:matrixMode/1*) is multiplied by this translation matrix, with the product replacing the current matrix, as if *gl:multMatrixd/1* were called with the following matrix for its argument:

(1 0 0 x 0 1 0 y 0 0 1 z 0 0 0 1)

If the matrix mode is either `?GL_MODELVIEW` or `?GL_PROJECTION`, all objects drawn after a call to `gl:translate` are translated.

Use *gl:pushMatrix/0* and *gl:pushMatrix/0* to save and restore the untranslated coordinate system.

See **external** documentation.

translatef(X, Y, Z) -> ok

Types:

X = float()
Y = float()
Z = float()

See *translated/3*

isList(List) -> 0 | 1

Types:

List = integer()

Determine if a name corresponds to a display list

`gl:isList` returns `?GL_TRUE` if `List` is the name of a display list and returns `?GL_FALSE` if it is not, or if an error occurs.

A name returned by *gl:genLists/1* , but not yet associated with a display list by calling *gl:newList/2* , is not the name of a display list.

See **external** documentation.

deleteLists(List, Range) -> ok

Types:

List = integer()

Range = integer()

Delete a contiguous group of display lists

`gl:deleteLists` causes a contiguous group of display lists to be deleted. `List` is the name of the first display list to be deleted, and `Range` is the number of display lists to delete. All display lists d with list<= d<= list+range-1 are deleted.

All storage locations allocated to the specified display lists are freed, and the names are available for reuse at a later time. Names within the range that do not have an associated display list are ignored. If `Range` is 0, nothing happens.

See **external** documentation.

genLists(Range) -> integer()

Types:

Range = integer()

Generate a contiguous set of empty display lists

`gl:genLists` has one argument, `Range` . It returns an integer `n` such that `Range` contiguous empty display lists, named n, n+1, ..., n+range-1, are created. If `Range` is 0, if there is no group of `Range` contiguous names available, or if any error is generated, no display lists are generated, and 0 is returned.

See **external** documentation.

newList(List, Mode) -> ok

Types:

List = integer()

Mode = enum()

Create or replace a display list

Display lists are groups of GL commands that have been stored for subsequent execution. Display lists are created with `gl:newList`. All subsequent commands are placed in the display list, in the order issued, until *gl:endList/0* is called.

`gl:newList` has two arguments. The first argument, `List` , is a positive integer that becomes the unique name for the display list. Names can be created and reserved with *gl:genLists/1* and tested for uniqueness with *gl:isList/1* . The second argument, `Mode` , is a symbolic constant that can assume one of two values:

`?GL_COMPILE`: Commands are merely compiled.

?GL_COMPILE_AND_EXECUTE: Commands are executed as they are compiled into the display list.

Certain commands are not compiled into the display list but are executed immediately, regardless of the display-list mode. These commands are *gl:areTexturesResident/1* , *gl:colorPointer/4* , *gl:deleteLists/2* , *gl:deleteTextures/1* , *gl:enableClientState/1* , *gl:edgeFlagPointer/2* , *gl:enableClientState/1* , *gl:feedbackBuffer/3* , *gl:finish/0* , *gl:flush/0* , *gl:genLists/1* , *gl:genTextures/1* , *gl:indexPointer/3* , *gl:interleavedArrays/3* , *gl:isEnabled/1* , *gl:isList/1* , *gl:isTexture/1* , *gl:normalPointer/3* , *gl:pushClientAttrib/1* , *gl:pixelStoref/2* , *gl:pushClientAttrib/1* , *gl:readPixels/7* , *gl:renderMode/1* , *gl:selectBuffer/2* , *gl:texCoordPointer/4* , *gl:vertexPointer/4* , and all of the *gl:getBooleanv/1* commands.

Similarly, *gl:texImage1D/8* , *gl:texImage2D/9* , and *gl:texImage3D/10* are executed immediately and not compiled into the display list when their first argument is ?GL_PROXY_TEXTURE_1D, ?GL_PROXY_TEXTURE_1D, or ?GL_PROXY_TEXTURE_3D , respectively.

When the ARB_imaging extension is supported, *gl:histogram/4* executes immediately when its argument is ?GL_PROXY_HISTOGRAM. Similarly, *gl:colorTable/6* executes immediately when its first argument is ?GL_PROXY_COLOR_TABLE, ?GL_PROXY_POST_CONVOLUTION_COLOR_TABLE , or ?GL_PROXY_POST_COLOR_MATRIX_COLOR_TABLE.

For OpenGL versions 1.3 and greater, or when the ARB_multitexture extension is supported, *gl:clientActiveTexture/1* is not compiled into display lists, but executed immediately.

When *gl:endList/0* is encountered, the display-list definition is completed by associating the list with the unique name List (specified in the gl:newList command). If a display list with name List already exists, it is replaced only when *gl:endList/0* is called.

See **external** documentation.

endList() -> ok

glBeginList

See **external** documentation.

callList(List) -> ok

Types:

```
List = integer()
```

Execute a display list

gl:callList causes the named display list to be executed. The commands saved in the display list are executed in order, just as if they were called without using a display list. If List has not been defined as a display list, gl:callList is ignored.

gl:callList can appear inside a display list. To avoid the possibility of infinite recursion resulting from display lists calling one another, a limit is placed on the nesting level of display lists during display-list execution. This limit is at least 64, and it depends on the implementation.

GL state is not saved and restored across a call to gl:callList. Thus, changes made to GL state during the execution of a display list remain after execution of the display list is completed. Use *gl:pushAttrib/1* , *gl:pushAttrib/1* , *gl:pushMatrix/0* , and *gl:pushMatrix/0* to preserve GL state across gl:callList calls.

See **external** documentation.

callLists(Lists) -> ok

Types:

```
Lists = [integer()]
```

Execute a list of display lists

`gl:callLists` causes each display list in the list of names passed as `Lists` to be executed. As a result, the commands saved in each display list are executed in order, just as if they were called without using a display list. Names of display lists that have not been defined are ignored.

`gl:callLists` provides an efficient means for executing more than one display list. `Type` allows lists with various name formats to be accepted. The formats are as follows:

`?GL_BYTE`: `Lists` is treated as an array of signed bytes, each in the range -128 through 127.

`?GL_UNSIGNED_BYTE`: `Lists` is treated as an array of unsigned bytes, each in the range 0 through 255.

`?GL_SHORT`: `Lists` is treated as an array of signed two-byte integers, each in the range -32768 through 32767.

`?GL_UNSIGNED_SHORT`: `Lists` is treated as an array of unsigned two-byte integers, each in the range 0 through 65535.

`?GL_INT`: `Lists` is treated as an array of signed four-byte integers.

`?GL_UNSIGNED_INT`: `Lists` is treated as an array of unsigned four-byte integers.

`?GL_FLOAT`: `Lists` is treated as an array of four-byte floating-point values.

`?GL_2_BYTES`: `Lists` is treated as an array of unsigned bytes. Each pair of bytes specifies a single display-list name. The value of the pair is computed as 256 times the unsigned value of the first byte plus the unsigned value of the second byte.

`?GL_3_BYTES`: `Lists` is treated as an array of unsigned bytes. Each triplet of bytes specifies a single display-list name. The value of the triplet is computed as 65536 times the unsigned value of the first byte, plus 256 times the unsigned value of the second byte, plus the unsigned value of the third byte.

`?GL_4_BYTES`: `Lists` is treated as an array of unsigned bytes. Each quadruplet of bytes specifies a single display-list name. The value of the quadruplet is computed as 16777216 times the unsigned value of the first byte, plus 65536 times the unsigned value of the second byte, plus 256 times the unsigned value of the third byte, plus the unsigned value of the fourth byte.

The list of display-list names is not null-terminated. Rather, `N` specifies how many names are to be taken from `Lists` .

An additional level of indirection is made available with the *gl:listBase/1* command, which specifies an unsigned offset that is added to each display-list name specified in `Lists` before that display list is executed.

`gl:callLists` can appear inside a display list. To avoid the possibility of infinite recursion resulting from display lists calling one another, a limit is placed on the nesting level of display lists during display-list execution. This limit must be at least 64, and it depends on the implementation.

GL state is not saved and restored across a call to `gl:callLists`. Thus, changes made to GL state during the execution of the display lists remain after execution is completed. Use *gl:pushAttrib/1* , *gl:pushAttrib/1* , *gl:pushMatrix/0* , and *gl:pushMatrix/0* to preserve GL state across `gl:callLists` calls.

See **external** documentation.

`listBase(Base) -> ok`

Types:

`Base = integer()`

set the display-list base for

gl:callLists/1

gl:callLists/1 specifies an array of offsets. Display-list names are generated by adding `Base` to each offset. Names that reference valid display lists are executed; the others are ignored.

See **external** documentation.

begin(Mode) -> ok

Types:

Mode = enum()

Delimit the vertices of a primitive or a group of like primitives

gl:'begin' and *gl:'begin'/1* delimit the vertices that define a primitive or a group of like primitives. gl:'begin' accepts a single argument that specifies in which of ten ways the vertices are interpreted. Taking n as an integer count starting at one, and N as the total number of vertices specified, the interpretations are as follows:

?GL_POINTS: Treats each vertex as a single point. Vertex n defines point n. N points are drawn.

?GL_LINES: Treats each pair of vertices as an independent line segment. Vertices 2 n-1 and 2 n define line n. N/2 lines are drawn.

?GL_LINE_STRIP: Draws a connected group of line segments from the first vertex to the last. Vertices n and n+1 define line n. N-1 lines are drawn.

?GL_LINE_LOOP: Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices n and n+1 define line n. The last line, however, is defined by vertices N and 1. N lines are drawn.

?GL_TRIANGLES: Treats each triplet of vertices as an independent triangle. Vertices 3 n-2, 3 n-1, and 3 n define triangle n. N/3 triangles are drawn.

?GL_TRIANGLE_STRIP: Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. For odd n, vertices n, n+1, and n+2 define triangle n. For even n, vertices n+1, n, and n +2 define triangle n. N-2 triangles are drawn.

?GL_TRIANGLE_FAN: Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. Vertices 1, n+1, and n+2 define triangle n. N-2 triangles are drawn.

?GL_QUADS: Treats each group of four vertices as an independent quadrilateral. Vertices 4 n-3, 4 n-2, 4 n-1, and 4 n define quadrilateral n. N/4 quadrilaterals are drawn.

?GL_QUAD_STRIP: Draws a connected group of quadrilaterals. One quadrilateral is defined for each pair of vertices presented after the first pair. Vertices 2 n-1, 2 n, 2 n+2, and 2 n+1 define quadrilateral n. N/2-1 quadrilaterals are drawn. Note that the order in which vertices are used to construct a quadrilateral from strip data is different from that used with independent data.

?GL_POLYGON: Draws a single, convex polygon. Vertices 1 through N define this polygon.

Only a subset of GL commands can be used between gl:'begin' and *gl:'begin'/1* . The commands are *gl:vertex2d/2* , *gl:color3b/3* , *gl:secondaryColor3b/3* , *gl:indexd/1* , *gl:normal3b/3* , *gl:fogCoordf/1* , *gl:texCoord1d/1* , *gl:multiTexCoord1d/2* , *gl:vertexAttrib1d/2* , *gl:evalCoord1d/1* , *gl:evalPoint1/1* , *gl:arrayElement/1* , *gl:materialf/3* , and *gl:edgeFlag/1* . Also, it is acceptable to use *gl:callList/1* or *gl:callLists/1* to execute display lists that include only the preceding commands. If any other GL command is executed between gl:'begin' and *gl:'begin'/1* , the error flag is set and the command is ignored.

Regardless of the value chosen for Mode , there is no limit to the number of vertices that can be defined between gl:'begin' and *gl:'begin'/1* . Lines, triangles, quadrilaterals, and polygons that are incompletely specified are not drawn. Incomplete specification results when either too few vertices are provided to specify even a single primitive or when an incorrect multiple of vertices is specified. The incomplete primitive is ignored; the rest are drawn.

The minimum specification of vertices for each primitive is as follows: 1 for a point, 2 for a line, 3 for a triangle, 4 for a quadrilateral, and 3 for a polygon. Modes that require a certain multiple of vertices are ?GL_LINES (2), ?GL_TRIANGLES (3), ?GL_QUADS (4), and ?GL_QUAD_STRIP (2).

See **external** documentation.

end() -> ok

See *'begin'/1*

vertex2d(X, Y) -> ok

Types:

X = float()
Y = float()

Specify a vertex

`gl:vertex` commands are used within *gl:'begin'/1* / *gl:'begin'/1* pairs to specify point, line, and polygon vertices. The current color, normal, texture coordinates, and fog coordinate are associated with the vertex when `gl:vertex` is called.

When only x and y are specified, z defaults to 0 and w defaults to 1. When x, y, and z are specified, w defaults to 1.

See **external** documentation.

vertex2f(X, Y) -> ok

Types:

X = float()
Y = float()

See *vertex2d/2*

vertex2i(X, Y) -> ok

Types:

X = integer()
Y = integer()

See *vertex2d/2*

vertex2s(X, Y) -> ok

Types:

X = integer()
Y = integer()

See *vertex2d/2*

vertex3d(X, Y, Z) -> ok

Types:

X = float()
Y = float()
Z = float()

See *vertex2d/2*

vertex3f(X, Y, Z) -> ok

Types:

X = float()
Y = float()

Z = float()

See *vertex2d/2*

vertex3i(X, Y, Z) -> ok

Types:

X = integer()
Y = integer()
Z = integer()

See *vertex2d/2*

vertex3s(X, Y, Z) -> ok

Types:

X = integer()
Y = integer()
Z = integer()

See *vertex2d/2*

vertex4d(X, Y, Z, W) -> ok

Types:

X = float()
Y = float()
Z = float()
W = float()

See *vertex2d/2*

vertex4f(X, Y, Z, W) -> ok

Types:

X = float()
Y = float()
Z = float()
W = float()

See *vertex2d/2*

vertex4i(X, Y, Z, W) -> ok

Types:

X = integer()
Y = integer()
Z = integer()
W = integer()

See *vertex2d/2*

vertex4s(X, Y, Z, W) -> ok

Types:

```
X = integer()
Y = integer()
Z = integer()
W = integer()
```

See *vertex2d/2*

vertex2dv(V) -> ok

Types:

```
V = {X::float(), Y::float()}
```

Equivalent to *vertex2d(X, Y)*.

vertex2fv(V) -> ok

Types:

```
V = {X::float(), Y::float()}
```

Equivalent to *vertex2f(X, Y)*.

vertex2iv(V) -> ok

Types:

```
V = {X::integer(), Y::integer()}
```

Equivalent to *vertex2i(X, Y)*.

vertex2sv(V) -> ok

Types:

```
V = {X::integer(), Y::integer()}
```

Equivalent to *vertex2s(X, Y)*.

vertex3dv(V) -> ok

Types:

```
V = {X::float(), Y::float(), Z::float()}
```

Equivalent to *vertex3d(X, Y, Z)*.

vertex3fv(V) -> ok

Types:

```
V = {X::float(), Y::float(), Z::float()}
```

Equivalent to *vertex3f(X, Y, Z)*.

vertex3iv(V) -> ok

Types:

```
V = {X::integer(), Y::integer(), Z::integer()}
```

Equivalent to *vertex3i(X, Y, Z)*.

vertex3sv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer()}

Equivalent to *vertex3s(X, Y, Z)*.

vertex4dv(V) -> ok

Types:

V = {X::float(), Y::float(), Z::float(), W::float()}

Equivalent to *vertex4d(X, Y, Z, W)*.

vertex4fv(V) -> ok

Types:

V = {X::float(), Y::float(), Z::float(), W::float()}

Equivalent to *vertex4f(X, Y, Z, W)*.

vertex4iv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer(), W::integer()}

Equivalent to *vertex4i(X, Y, Z, W)*.

vertex4sv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer(), W::integer()}

Equivalent to *vertex4s(X, Y, Z, W)*.

normal3b(Nx, Ny, Nz) -> ok

Types:

Nx = integer()
Ny = integer()
Nz = integer()

Set the current normal vector

The current normal is set to the given coordinates whenever `gl:normal` is issued. Byte, short, or integer arguments are converted to floating-point format with a linear mapping that maps the most positive representable integer value to 1.0 and the most negative representable integer value to -1.0.

Normals specified with `gl:normal` need not have unit length. If `?GL_NORMALIZE` is enabled, then normals of any length specified with `gl:normal` are normalized after transformation. If `?GL_RESCALE_NORMAL` is enabled, normals are scaled by a scaling factor derived from the modelview matrix. `?GL_RESCALE_NORMAL` requires that the originally specified normals were of unit length, and that the modelview matrix contain only uniform scales for proper results. To enable and disable normalization, call *gl:enable/1* and *gl:enable/1* with either `?GL_NORMALIZE` or `?GL_RESCALE_NORMAL`. Normalization is initially disabled.

See **external** documentation.

normal3d(Nx, Ny, Nz) -> ok

Types:

Nx = float()
Ny = float()

Nz = float()

See *normal3b/3*

normal3f(Nx, Ny, Nz) -> ok

Types:

Nx = float()
Ny = float()
Nz = float()

See *normal3b/3*

normal3i(Nx, Ny, Nz) -> ok

Types:

Nx = integer()
Ny = integer()
Nz = integer()

See *normal3b/3*

normal3s(Nx, Ny, Nz) -> ok

Types:

Nx = integer()
Ny = integer()
Nz = integer()

See *normal3b/3*

normal3bv(V) -> ok

Types:

V = {Nx::integer(), Ny::integer(), Nz::integer()}

Equivalent to *normal3b(Nx, Ny, Nz)*.

normal3dv(V) -> ok

Types:

V = {Nx::float(), Ny::float(), Nz::float()}

Equivalent to *normal3d(Nx, Ny, Nz)*.

normal3fv(V) -> ok

Types:

V = {Nx::float(), Ny::float(), Nz::float()}

Equivalent to *normal3f(Nx, Ny, Nz)*.

normal3iv(V) -> ok

Types:

V = {Nx::integer(), Ny::integer(), Nz::integer()}

Equivalent to *normal3i(Nx, Ny, Nz)*.

normal3sv(V) -> ok

Types:

V = {Nx::integer(), Ny::integer(), Nz::integer()}

Equivalent to *normal3s(Nx, Ny, Nz)*.

indexd(C) -> ok

Types:

C = float()

Set the current color index

`gl:index` updates the current (single-valued) color index. It takes one argument, the new value for the current color index.

The current index is stored as a floating-point value. Integer values are converted directly to floating-point values, with no special mapping. The initial value is 1.

Index values outside the representable range of the color index buffer are not clamped. However, before an index is dithered (if enabled) and written to the frame buffer, it is converted to fixed-point format. Any bits in the integer portion of the resulting fixed-point value that do not correspond to bits in the frame buffer are masked out.

See **external** documentation.

indexf(C) -> ok

Types:

C = float()

See *indexd/1*

indexi(C) -> ok

Types:

C = integer()

See *indexd/1*

indexs(C) -> ok

Types:

C = integer()

See *indexd/1*

indexub(C) -> ok

Types:

C = integer()

See *indexd/1*

indexdv(C) -> ok

Types:

C = {C::float()}

Equivalent to *indexd(C)*.

indexfv(C) -> ok

Types:

C = {C::float()}

Equivalent to *indexf(C)*.

indexiv(C) -> ok

Types:

C = {C::integer()}

Equivalent to *indexi(C)*.

indexsv(C) -> ok

Types:

C = {C::integer()}

Equivalent to *indexs(C)*.

indexubv(C) -> ok

Types:

C = {C::integer()}

Equivalent to *indexub(C)*.

color3b(Red, Green, Blue) -> ok

Types:

Red = integer()

Green = integer()

Blue = integer()

Set the current color

The GL stores both a current single-valued color index and a current four-valued RGBA color. `gl:color` sets a new four-valued RGBA color. `gl:color` has two major variants: `gl:color3` and `gl:color4`. `gl:color3` variants specify new red, green, and blue values explicitly and set the current alpha value to 1.0 (full intensity) implicitly. `gl:color4` variants specify all four color components explicitly.

`gl:color3b`, `gl:color4b`, `gl:color3s`, `gl:color4s`, `gl:color3i`, and `gl:color4i` take three or four signed byte, short, or long integers as arguments. When `v` is appended to the name, the color commands can take a pointer to an array of such values.

Current color values are stored in floating-point format, with unspecified mantissa and exponent sizes. Unsigned integer color components, when specified, are linearly mapped to floating-point values such that the largest representable value maps to 1.0 (full intensity), and 0 maps to 0.0 (zero intensity). Signed integer color components, when specified, are linearly mapped to floating-point values such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. (Note that this mapping does not convert 0 precisely to 0.0.) Floating-point values are mapped directly.

Neither floating-point nor signed integer values are clamped to the range [0 1] before the current color is updated. However, color components are clamped to this range before they are interpolated or written into a color buffer.

See **external** documentation.

color3d(Red, Green, Blue) -> ok

Types:

Red = float()
Green = float()
Blue = float()

See *color3b/3*

color3f(Red, Green, Blue) -> ok

Types:

Red = float()
Green = float()
Blue = float()

See *color3b/3*

color3i(Red, Green, Blue) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()

See *color3b/3*

color3s(Red, Green, Blue) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()

See *color3b/3*

color3ub(Red, Green, Blue) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()

See *color3b/3*

color3ui(Red, Green, Blue) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()

See *color3b/3*

color3us(Red, Green, Blue) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()

See *color3b/3*

color4b(Red, Green, Blue, Alpha) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()
Alpha = integer()

See *color3b/3*

color4d(Red, Green, Blue, Alpha) -> ok

Types:

Red = float()
Green = float()
Blue = float()
Alpha = float()

See *color3b/3*

color4f(Red, Green, Blue, Alpha) -> ok

Types:

Red = float()
Green = float()
Blue = float()
Alpha = float()

See *color3b/3*

color4i(Red, Green, Blue, Alpha) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()
Alpha = integer()

See *color3b/3*

color4s(Red, Green, Blue, Alpha) -> ok

Types:

Red = integer()
Green = integer()

Blue = integer()
Alpha = integer()

See *color3b/3*

color4ub(Red, Green, Blue, Alpha) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()
Alpha = integer()

See *color3b/3*

color4ui(Red, Green, Blue, Alpha) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()
Alpha = integer()

See *color3b/3*

color4us(Red, Green, Blue, Alpha) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()
Alpha = integer()

See *color3b/3*

color3bv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *color3b(Red, Green, Blue)*.

color3dv(V) -> ok

Types:

V = {Red::float(), Green::float(), Blue::float()}

Equivalent to *color3d(Red, Green, Blue)*.

color3fv(V) -> ok

Types:

V = {Red::float(), Green::float(), Blue::float()}

Equivalent to *color3f(Red, Green, Blue)*.

color3iv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *color3i(Red, Green, Blue)*.

color3sv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *color3s(Red, Green, Blue)*.

color3ubv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *color3ub(Red, Green, Blue)*.

color3uiv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *color3ui(Red, Green, Blue)*.

color3usv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *color3us(Red, Green, Blue)*.

color4bv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer(), Alpha::integer()}

Equivalent to *color4b(Red, Green, Blue, Alpha)*.

color4dv(V) -> ok

Types:

V = {Red::float(), Green::float(), Blue::float(), Alpha::float()}

Equivalent to *color4d(Red, Green, Blue, Alpha)*.

color4fv(V) -> ok

Types:

V = {Red::float(), Green::float(), Blue::float(), Alpha::float()}

Equivalent to *color4f(Red, Green, Blue, Alpha)*.

color4iv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer(), Alpha::integer()}

Equivalent to *color4i(Red, Green, Blue, Alpha)*.

color4sv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer(), Alpha::integer()}

Equivalent to *color4s(Red, Green, Blue, Alpha)*.

color4ubv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer(), Alpha::integer()}

Equivalent to *color4ub(Red, Green, Blue, Alpha)*.

color4uiv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer(), Alpha::integer()}

Equivalent to *color4ui(Red, Green, Blue, Alpha)*.

color4usv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer(), Alpha::integer()}

Equivalent to *color4us(Red, Green, Blue, Alpha)*.

texCoord1d(S) -> ok

Types:

S = float()

Set the current texture coordinates

`gl:texCoord` specifies texture coordinates in one, two, three, or four dimensions. `gl:texCoord1` sets the current texture coordinates to (s 0 0 1); a call to `gl:texCoord2` sets them to (s t 0 1). Similarly, `gl:texCoord3` specifies the texture coordinates as (s t r 1), and `gl:texCoord4` defines all four components explicitly as (s t r q).

The current texture coordinates are part of the data that is associated with each vertex and with the current raster position. Initially, the values for `s`, `t`, `r` , and `q` are (0, 0, 0, 1).

See **external** documentation.

texCoord1f(S) -> ok

Types:

S = float()

See *texCoord1d/1*

texCoord1i(S) -> ok

Types:

S = integer()

See *texCoord1d/1*

texCoord1s(S) -> ok

Types:

S = integer()

See *texCoord1d/1*

texCoord2d(S, T) -> ok

Types:

S = float()

T = float()

See *texCoord1d/1*

texCoord2f(S, T) -> ok

Types:

S = float()

T = float()

See *texCoord1d/1*

texCoord2i(S, T) -> ok

Types:

S = integer()

T = integer()

See *texCoord1d/1*

texCoord2s(S, T) -> ok

Types:

S = integer()

T = integer()

See *texCoord1d/1*

texCoord3d(S, T, R) -> ok

Types:

S = float()

T = float()

R = float()

See *texCoord1d/1*

texCoord3f(S, T, R) -> ok

Types:

S = float()

T = float()

R = float()

See *texCoord1d/1*

texCoord3i(S, T, R) -> ok

Types:

S = integer()
T = integer()
R = integer()

See *texCoord1d/1*

texCoord3s(S, T, R) -> ok

Types:

S = integer()
T = integer()
R = integer()

See *texCoord1d/1*

texCoord4d(S, T, R, Q) -> ok

Types:

S = float()
T = float()
R = float()
Q = float()

See *texCoord1d/1*

texCoord4f(S, T, R, Q) -> ok

Types:

S = float()
T = float()
R = float()
Q = float()

See *texCoord1d/1*

texCoord4i(S, T, R, Q) -> ok

Types:

S = integer()
T = integer()
R = integer()
Q = integer()

See *texCoord1d/1*

texCoord4s(S, T, R, Q) -> ok

Types:

S = integer()
T = integer()
R = integer()

Q = integer()

See *texCoord1d/1*

texCoord1dv(V) -> ok

Types:

V = {S::float()}

Equivalent to *texCoord1d(S)*.

texCoord1fv(V) -> ok

Types:

V = {S::float()}

Equivalent to *texCoord1f(S)*.

texCoord1iv(V) -> ok

Types:

V = {S::integer()}

Equivalent to *texCoord1i(S)*.

texCoord1sv(V) -> ok

Types:

V = {S::integer()}

Equivalent to *texCoord1s(S)*.

texCoord2dv(V) -> ok

Types:

V = {S::float(), T::float()}

Equivalent to *texCoord2d(S, T)*.

texCoord2fv(V) -> ok

Types:

V = {S::float(), T::float()}

Equivalent to *texCoord2f(S, T)*.

texCoord2iv(V) -> ok

Types:

V = {S::integer(), T::integer()}

Equivalent to *texCoord2i(S, T)*.

texCoord2sv(V) -> ok

Types:

V = {S::integer(), T::integer()}

Equivalent to *texCoord2s(S, T)*.

texCoord3dv(V) -> ok

Types:

V = {S::float(), T::float(), R::float()}

Equivalent to *texCoord3d(S, T, R)*.

texCoord3fv(V) -> ok

Types:

V = {S::float(), T::float(), R::float()}

Equivalent to *texCoord3f(S, T, R)*.

texCoord3iv(V) -> ok

Types:

V = {S::integer(), T::integer(), R::integer()}

Equivalent to *texCoord3i(S, T, R)*.

texCoord3sv(V) -> ok

Types:

V = {S::integer(), T::integer(), R::integer()}

Equivalent to *texCoord3s(S, T, R)*.

texCoord4dv(V) -> ok

Types:

V = {S::float(), T::float(), R::float(), Q::float()}

Equivalent to *texCoord4d(S, T, R, Q)*.

texCoord4fv(V) -> ok

Types:

V = {S::float(), T::float(), R::float(), Q::float()}

Equivalent to *texCoord4f(S, T, R, Q)*.

texCoord4iv(V) -> ok

Types:

V = {S::integer(), T::integer(), R::integer(), Q::integer()}

Equivalent to *texCoord4i(S, T, R, Q)*.

texCoord4sv(V) -> ok

Types:

V = {S::integer(), T::integer(), R::integer(), Q::integer()}

Equivalent to *texCoord4s(S, T, R, Q)*.

rasterPos2d(X, Y) -> ok

Types:

X = float()

Y = float()

Specify the raster position for pixel operations

The GL maintains a 3D position in window coordinates. This position, called the raster position, is used to position pixel and bitmap write operations. It is maintained with subpixel accuracy. See *gl:bitmap/7* , *gl:drawPixels/5* , and *gl:copyPixels/5* .

The current raster position consists of three window coordinates (x, y, z), a clip coordinate value (w), an eye coordinate distance, a valid bit, and associated color data and texture coordinates. The w coordinate is a clip coordinate, because w is not projected to window coordinates. `gl:rasterPos4` specifies object coordinates x, y, z, and w explicitly. `gl:rasterPos3` specifies object coordinate x, y, and z explicitly, while w is implicitly set to 1. `gl:rasterPos2` uses the argument values for x and y while implicitly setting z and w to 0 and 1.

The object coordinates presented by `gl:rasterPos` are treated just like those of a *gl:vertex2d/2* command: They are transformed by the current modelview and projection matrices and passed to the clipping stage. If the vertex is not culled, then it is projected and scaled to window coordinates, which become the new current raster position, and the `?GL_CURRENT_RASTER_POSITION_VALID` flag is set. If the vertex `is` culled, then the valid bit is cleared and the current raster position and associated color and texture coordinates are undefined.

The current raster position also includes some associated color data and texture coordinates. If lighting is enabled, then `?GL_CURRENT_RASTER_COLOR` (in RGBA mode) or `?GL_CURRENT_RASTER_INDEX` (in color index mode) is set to the color produced by the lighting calculation (see *gl:lightf/3* , *gl:lightModelf/2* , and *gl:shadeModel/1*). If lighting is disabled, current color (in RGBA mode, state variable `?GL_CURRENT_COLOR`) or color index (in color index mode, state variable `?GL_CURRENT_INDEX`) is used to update the current raster color. `?GL_CURRENT_RASTER_SECONDARY_COLOR` (in RGBA mode) is likewise updated.

Likewise, `?GL_CURRENT_RASTER_TEXTURE_COORDS` is updated as a function of `?GL_CURRENT_TEXTURE_COORDS` , based on the texture matrix and the texture generation functions (see *gl:texGend/3*). Finally, the distance from the origin of the eye coordinate system to the vertex as transformed by only the modelview matrix replaces `?GL_CURRENT_RASTER_DISTANCE`.

Initially, the current raster position is (0, 0, 0, 1), the current raster distance is 0, the valid bit is set, the associated RGBA color is (1, 1, 1, 1), the associated color index is 1, and the associated texture coordinates are (0, 0, 0, 1). In RGBA mode, `?GL_CURRENT_RASTER_INDEX` is always 1; in color index mode, the current raster RGBA color always maintains its initial value.

See **external** documentation.

rasterPos2f(X, Y) -> ok

Types:

X = float()

Y = float()

See *rasterPos2d/2*

rasterPos2i(X, Y) -> ok

Types:

X = integer()

Y = integer()

See *rasterPos2d/2*

rasterPos2s(X, Y) -> ok

Types:

```
X = integer()
Y = integer()
```

See *rasterPos2d/2*

rasterPos3d(X, Y, Z) -> ok

Types:

```
X = float()
Y = float()
Z = float()
```

See *rasterPos2d/2*

rasterPos3f(X, Y, Z) -> ok

Types:

```
X = float()
Y = float()
Z = float()
```

See *rasterPos2d/2*

rasterPos3i(X, Y, Z) -> ok

Types:

```
X = integer()
Y = integer()
Z = integer()
```

See *rasterPos2d/2*

rasterPos3s(X, Y, Z) -> ok

Types:

```
X = integer()
Y = integer()
Z = integer()
```

See *rasterPos2d/2*

rasterPos4d(X, Y, Z, W) -> ok

Types:

```
X = float()
Y = float()
Z = float()
W = float()
```

See *rasterPos2d/2*

rasterPos4f(X, Y, Z, W) -> ok

Types:

```
X = float()
```

Y = float()
Z = float()
W = float()

See *rasterPos2d/2*

rasterPos4i(X, Y, Z, W) -> ok

Types:

X = integer()
Y = integer()
Z = integer()
W = integer()

See *rasterPos2d/2*

rasterPos4s(X, Y, Z, W) -> ok

Types:

X = integer()
Y = integer()
Z = integer()
W = integer()

See *rasterPos2d/2*

rasterPos2dv(V) -> ok

Types:

V = {X::float(), Y::float()}

Equivalent to *rasterPos2d(X, Y)*.

rasterPos2fv(V) -> ok

Types:

V = {X::float(), Y::float()}

Equivalent to *rasterPos2f(X, Y)*.

rasterPos2iv(V) -> ok

Types:

V = {X::integer(), Y::integer()}

Equivalent to *rasterPos2i(X, Y)*.

rasterPos2sv(V) -> ok

Types:

V = {X::integer(), Y::integer()}

Equivalent to *rasterPos2s(X, Y)*.

rasterPos3dv(V) -> ok

Types:

V = {X::float(), Y::float(), Z::float()}

Equivalent to *rasterPos3d(X, Y, Z)*.

rasterPos3fv(V) -> ok

Types:

V = {X::float(), Y::float(), Z::float()}

Equivalent to *rasterPos3f(X, Y, Z)*.

rasterPos3iv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer()}

Equivalent to *rasterPos3i(X, Y, Z)*.

rasterPos3sv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer()}

Equivalent to *rasterPos3s(X, Y, Z)*.

rasterPos4dv(V) -> ok

Types:

V = {X::float(), Y::float(), Z::float(), W::float()}

Equivalent to *rasterPos4d(X, Y, Z, W)*.

rasterPos4fv(V) -> ok

Types:

V = {X::float(), Y::float(), Z::float(), W::float()}

Equivalent to *rasterPos4f(X, Y, Z, W)*.

rasterPos4iv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer(), W::integer()}

Equivalent to *rasterPos4i(X, Y, Z, W)*.

rasterPos4sv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer(), W::integer()}

Equivalent to *rasterPos4s(X, Y, Z, W)*.

rectd(X1, Y1, X2, Y2) -> ok

Types:

X1 = float()
Y1 = float()
X2 = float()

Y2 = float()

Draw a rectangle

`gl:rect` supports efficient specification of rectangles as two corner points. Each rectangle command takes four arguments, organized either as two consecutive pairs of (x y) coordinates or as two pointers to arrays, each containing an (x y) pair. The resulting rectangle is defined in the z= 0 plane.

`gl:rect`(`X1` , `Y1` , `X2` , `Y2`) is exactly equivalent to the following sequence: glBegin(`?GL_POLYGON`); glVertex2(`X1` , `Y1`); glVertex2(`X2` , `Y1`); glVertex2(`X2` , `Y2`); glVertex2(`X1` , `Y2`); glEnd(); Note that if the second vertex is above and to the right of the first vertex, the rectangle is constructed with a counterclockwise winding.

See **external** documentation.

rectf(X1, Y1, X2, Y2) -> ok

Types:

X1 = float()
Y1 = float()
X2 = float()
Y2 = float()

See *rectd/4*

recti(X1, Y1, X2, Y2) -> ok

Types:

X1 = integer()
Y1 = integer()
X2 = integer()
Y2 = integer()

See *rectd/4*

rects(X1, Y1, X2, Y2) -> ok

Types:

X1 = integer()
Y1 = integer()
X2 = integer()
Y2 = integer()

See *rectd/4*

rectdv(V1, V2) -> ok

Types:

V1 = {float(), float()}
V2 = {float(), float()}

See *rectd/4*

rectfv(V1, V2) -> ok

Types:

V1 = {float(), float()}

V2 = {float(), float()}

See *rectd/4*

rectiv(V1, V2) -> ok

Types:

V1 = {integer(), integer()}
V2 = {integer(), integer()}

See *rectd/4*

rectsv(V1, V2) -> ok

Types:

V1 = {integer(), integer()}
V2 = {integer(), integer()}

See *rectd/4*

vertexPointer(Size, Type, Stride, Ptr) -> ok

Types:

Size = integer()
Type = enum()
Stride = integer()
Ptr = offset() | mem()

Define an array of vertex data

`gl:vertexPointer` specifies the location and data format of an array of vertex coordinates to use when rendering. `Size` specifies the number of coordinates per vertex, and must be 2, 3, or 4. `Type` specifies the data type of each coordinate, and `Stride` specifies the byte stride from one vertex to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays. (Single-array storage may be more efficient on some implementations; see *gl:interleavedArrays/3* .)

If a non-zero named buffer object is bound to the `?GL_ARRAY_BUFFER` target (see *gl:bindBuffer/2*) while a vertex array is specified, `Pointer` is treated as a byte offset into the buffer object's data store. Also, the buffer object binding (`?GL_ARRAY_BUFFER_BINDING`) is saved as vertex array client-side state (`?GL_VERTEX_ARRAY_BUFFER_BINDING`).

When a vertex array is specified, `Size` , `Type` , `Stride` , and `Pointer` are saved as client-side state, in addition to the current vertex array buffer object binding.

To enable and disable the vertex array, call *gl:enableClientState/1* and *gl:enableClientState/1* with the argument `?GL_VERTEX_ARRAY`. If enabled, the vertex array is used when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see `glMultiDrawElements` , or *gl:drawRangeElements/6* is called.

See **external** documentation.

normalPointer(Type, Stride, Ptr) -> ok

Types:

Type = enum()
Stride = integer()
Ptr = offset() | mem()

Define an array of normals

gl:normalPointer specifies the location and data format of an array of normals to use when rendering. Type specifies the data type of each normal coordinate, and Stride specifies the byte stride from one normal to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays. (Single-array storage may be more efficient on some implementations; see *gl:interleavedArrays/3* .)

If a non-zero named buffer object is bound to the ?GL_ARRAY_BUFFER target (see *gl:bindBuffer/2*) while a normal array is specified, Pointer is treated as a byte offset into the buffer object's data store. Also, the buffer object binding (?GL_ARRAY_BUFFER_BINDING) is saved as normal vertex array client-side state (?GL_NORMAL_ARRAY_BUFFER_BINDING).

When a normal array is specified, Type , Stride , and Pointer are saved as client-side state, in addition to the current vertex array buffer object binding.

To enable and disable the normal array, call *gl:enableClientState/1* and *gl:enableClientState/1* with the argument ?GL_NORMAL_ARRAY. If enabled, the normal array is used when *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see glMultiDrawElements, *gl:drawRangeElements/6* , or *gl:arrayElement/1* is called.

See **external** documentation.

colorPointer(Size, Type, Stride, Ptr) -> ok

Types:

Size = integer()
Type = enum()
Stride = integer()
Ptr = offset() | mem()

Define an array of colors

gl:colorPointer specifies the location and data format of an array of color components to use when rendering. Size specifies the number of components per color, and must be 3 or 4. Type specifies the data type of each color component, and Stride specifies the byte stride from one color to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays. (Single-array storage may be more efficient on some implementations; see *gl:interleavedArrays/3* .)

If a non-zero named buffer object is bound to the ?GL_ARRAY_BUFFER target (see *gl:bindBuffer/2*) while a color array is specified, Pointer is treated as a byte offset into the buffer object's data store. Also, the buffer object binding (?GL_ARRAY_BUFFER_BINDING) is saved as color vertex array client-side state (?GL_COLOR_ARRAY_BUFFER_BINDING).

When a color array is specified, Size , Type , Stride , and Pointer are saved as client-side state, in addition to the current vertex array buffer object binding.

To enable and disable the color array, call *gl:enableClientState/1* and *gl:enableClientState/1* with the argument ?GL_COLOR_ARRAY. If enabled, the color array is used when *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see glMultiDrawElements, *gl:drawRangeElements/6* , or *gl:arrayElement/1* is called.

See **external** documentation.

indexPointer(Type, Stride, Ptr) -> ok

Types:

Type = enum()
Stride = integer()
Ptr = offset() | mem()

Define an array of color indexes

gl:indexPointer specifies the location and data format of an array of color indexes to use when rendering. Type specifies the data type of each color index and Stride specifies the byte stride from one color index to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays.

If a non-zero named buffer object is bound to the ?GL_ARRAY_BUFFER target (see *gl:bindBuffer/2*) while a color index array is specified, Pointer is treated as a byte offset into the buffer object's data store. Also, the buffer object binding (?GL_ARRAY_BUFFER_BINDING) is saved as color index vertex array client-side state (?GL_INDEX_ARRAY_BUFFER_BINDING).

When a color index array is specified, Type , Stride , and Pointer are saved as client-side state, in addition to the current vertex array buffer object binding.

To enable and disable the color index array, call *gl:enableClientState/1* and *gl:enableClientState/1* with the argument ?GL_INDEX_ARRAY. If enabled, the color index array is used when *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see glMultiDrawElements , *gl:drawRangeElements/6* , or *gl:arrayElement/1* is called.

See **external** documentation.

texCoordPointer(Size, Type, Stride, Ptr) -> ok

Types:

```
Size = integer()
Type = enum()
Stride = integer()
Ptr = offset() | mem()
```

Define an array of texture coordinates

gl:texCoordPointer specifies the location and data format of an array of texture coordinates to use when rendering. Size specifies the number of coordinates per texture coordinate set, and must be 1, 2, 3, or 4. Type specifies the data type of each texture coordinate, and Stride specifies the byte stride from one texture coordinate set to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays. (Single-array storage may be more efficient on some implementations; see *gl:interleavedArrays/3* .)

If a non-zero named buffer object is bound to the ?GL_ARRAY_BUFFER target (see *gl:bindBuffer/2*) while a texture coordinate array is specified, Pointer is treated as a byte offset into the buffer object's data store. Also, the buffer object binding (?GL_ARRAY_BUFFER_BINDING) is saved as texture coordinate vertex array client-side state (?GL_TEXTURE_COORD_ARRAY_BUFFER_BINDING).

When a texture coordinate array is specified, Size , Type , Stride , and Pointer are saved as client-side state, in addition to the current vertex array buffer object binding.

To enable and disable a texture coordinate array, call *gl:enableClientState/1* and *gl:enableClientState/1* with the argument ?GL_TEXTURE_COORD_ARRAY. If enabled, the texture coordinate array is used when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see glMultiDrawElements, or *gl:drawRangeElements/6* is called.

See **external** documentation.

edgeFlagPointer(Stride, Ptr) -> ok

Types:

```
Stride = integer()
Ptr = offset() | mem()
```

Define an array of edge flags

gl:edgeFlagPointer specifies the location and data format of an array of boolean edge flags to use when rendering. Stride specifies the byte stride from one edge flag to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays.

If a non-zero named buffer object is bound to the ?GL_ARRAY_BUFFER target (see *gl:bindBuffer/2*) while an edge flag array is specified, Pointer is treated as a byte offset into the buffer object's data store. Also, the buffer object binding (?GL_ARRAY_BUFFER_BINDING) is saved as edge flag vertex array client-side state (?GL_EDGE_FLAG_ARRAY_BUFFER_BINDING).

When an edge flag array is specified, Stride and Pointer are saved as client-side state, in addition to the current vertex array buffer object binding.

To enable and disable the edge flag array, call *gl:enableClientState/1* and *gl:enableClientState/1* with the argument ?GL_EDGE_FLAG_ARRAY. If enabled, the edge flag array is used when *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see glMultiDrawElements , *gl:drawRangeElements/6* , or *gl:arrayElement/1* is called.

See **external** documentation.

arrayElement(I) -> ok

Types:

I = integer()

Render a vertex using the specified vertex array element

gl:arrayElement commands are used within *gl:'begin'/1* / *gl:'begin'/1* pairs to specify vertex and attribute data for point, line, and polygon primitives. If ?GL_VERTEX_ARRAY is enabled when gl:arrayElement is called, a single vertex is drawn, using vertex and attribute data taken from location I of the enabled arrays. If ?GL_VERTEX_ARRAY is not enabled, no drawing occurs but the attributes corresponding to the enabled arrays are modified.

Use gl:arrayElement to construct primitives by indexing vertex data, rather than by streaming through arrays of data in first-to-last order. Because each call specifies only a single vertex, it is possible to explicitly specify per-primitive attributes such as a single normal for each triangle.

Changes made to array data between the execution of *gl:'begin'/1* and the corresponding execution of *gl:'begin'/1* may affect calls to gl:arrayElement that are made within the same *gl:'begin'/1* / *gl:'begin'/1* period in nonsequential ways. That is, a call to gl:arrayElement that precedes a change to array data may access the changed data, and a call that follows a change to array data may access original data.

See **external** documentation.

drawArrays(Mode, First, Count) -> ok

Types:

Mode = enum()

First = integer()

Count = integer()

Render primitives from array data

gl:drawArrays specifies multiple geometric primitives with very few subroutine calls. Instead of calling a GL procedure to pass each individual vertex, normal, texture coordinate, edge flag, or color, you can prespecify separate arrays of vertices, normals, and colors and use them to construct a sequence of primitives with a single call to gl:drawArrays .

When gl:drawArrays is called, it uses Count sequential elements from each enabled array to construct a sequence of geometric primitives, beginning with element First . Mode specifies what kind of primitives are constructed and how the array elements construct those primitives.

Vertex attributes that are modified by `gl:drawArrays` have an unspecified value after `gl:drawArrays` returns. Attributes that aren't modified remain well defined.

See **external** documentation.

`drawElements(Mode, Count, Type, Indices) -> ok`

Types:

```
Mode = enum()
Count = integer()
Type = enum()
Indices = offset() | mem()
```

Render primitives from array data

`gl:drawElements` specifies multiple geometric primitives with very few subroutine calls. Instead of calling a GL function to pass each individual vertex, normal, texture coordinate, edge flag, or color, you can prespecify separate arrays of vertices, normals, and so on, and use them to construct a sequence of primitives with a single call to `gl:drawElements` .

When `gl:drawElements` is called, it uses `Count` sequential elements from an enabled array, starting at `Indices` to construct a sequence of geometric primitives. `Mode` specifies what kind of primitives are constructed and how the array elements construct these primitives. If more than one array is enabled, each is used.

Vertex attributes that are modified by `gl:drawElements` have an unspecified value after `gl:drawElements` returns. Attributes that aren't modified maintain their previous values.

See **external** documentation.

`interleavedArrays(Format, Stride, Pointer) -> ok`

Types:

```
Format = enum()
Stride = integer()
Pointer = offset() | mem()
```

Simultaneously specify and enable several interleaved arrays

`gl:interleavedArrays` lets you specify and enable individual color, normal, texture and vertex arrays whose elements are part of a larger aggregate array element. For some implementations, this is more efficient than specifying the arrays separately.

If `Stride` is 0, the aggregate elements are stored consecutively. Otherwise, `Stride` bytes occur between the beginning of one aggregate array element and the beginning of the next aggregate array element.

`Format` serves as a `key` describing the extraction of individual arrays from the aggregate array. If `Format` contains a T, then texture coordinates are extracted from the interleaved array. If C is present, color values are extracted. If N is present, normal coordinates are extracted. Vertex coordinates are always extracted.

The digits 2, 3, and 4 denote how many values are extracted. F indicates that values are extracted as floating-point values. Colors may also be extracted as 4 unsigned bytes if 4UB follows the C. If a color is extracted as 4 unsigned bytes, the vertex array element which follows is located at the first possible floating-point aligned address.

See **external** documentation.

`shadeModel(Mode) -> ok`

Types:

```
Mode = enum()
```

Select flat or smooth shading

GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized, typically assigning different colors to each resulting pixel fragment. Flat shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive. In either case, the computed color of a vertex is the result of lighting if lighting is enabled, or it is the current color at the time the vertex was specified if lighting is disabled.

Flat and smooth shading are indistinguishable for points. Starting when *gl:'begin'/1* is issued and counting vertices and primitives from 1, the GL gives each flat-shaded line segment i the computed color of vertex i+1, its second vertex. Counting similarly from 1, the GL gives each flat-shaded polygon the computed color of the vertex listed in the following table. This is the last vertex to specify the polygon in all cases except single polygons, where the first vertex specifies the flat-shaded color.

`Primitive Type of Polygon` i`Vertex`
Single polygon (i== 1) 1
Triangle strip i+2
Triangle fan i+2
Independent triangle 3 i
Quad strip 2 i+2
Independent quad 4 i

Flat and smooth shading are specified by `gl:shadeModel` with `Mode` set to `?GL_FLAT` and `?GL_SMOOTH`, respectively.

See **external** documentation.

`lightf(Light, Pname, Param) -> ok`

Types:

```
Light = enum()
Pname = enum()
Param = float()
```

Set light source parameters

`gl:light` sets the values of individual light source parameters. `Light` names the light and is a symbolic name of the form `?GL_LIGHT` i, where i ranges from 0 to the value of `?GL_MAX_LIGHTS` - 1. `Pname` specifies one of ten light source parameters, again by symbolic name. `Params` is either a single value or a pointer to an array that contains the new values.

To enable and disable lighting calculation, call *gl:enable/1* and *gl:enable/1* with argument `?GL_LIGHTING`. Lighting is initially disabled. When it is enabled, light sources that are enabled contribute to the lighting calculation. Light source i is enabled and disabled using *gl:enable/1* and *gl:enable/1* with argument `?GL_LIGHT` i.

The ten light parameters are as follows:

`?GL_AMBIENT`: `Params` contains four integer or floating-point values that specify the ambient RGBA intensity of the light. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient light intensity is (0, 0, 0, 1).

`?GL_DIFFUSE`: `Params` contains four integer or floating-point values that specify the diffuse RGBA intensity of the light. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial value for `?GL_LIGHT0` is (1, 1, 1, 1); for other lights, the initial value is (0, 0, 0, 1).

`?GL_SPECULAR`: `Params` contains four integer or floating-point values that specify the specular RGBA intensity of the light. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most

negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial value for `?GL_LIGHT0` is (1, 1, 1, 1); for other lights, the initial value is (0, 0, 0, 1).

`?GL_POSITION`: `Params` contains four integer or floating-point values that specify the position of the light in homogeneous object coordinates. Both integer and floating-point values are mapped directly. Neither integer nor floating-point values are clamped.

The position is transformed by the modelview matrix when `gl:light` is called (just as if it were a point), and it is stored in eye coordinates. If the w component of the position is 0, the light is treated as a directional source. Diffuse and specular lighting calculations take the light's direction, but not its actual position, into account, and attenuation is disabled. Otherwise, diffuse and specular lighting calculations are based on the actual location of the light in eye coordinates, and attenuation is enabled. The initial position is (0, 0, 1, 0); thus, the initial light source is directional, parallel to, and in the direction of the -z axis.

`?GL_SPOT_DIRECTION`: `Params` contains three integer or floating-point values that specify the direction of the light in homogeneous object coordinates. Both integer and floating-point values are mapped directly. Neither integer nor floating-point values are clamped.

The spot direction is transformed by the upper 3x3 of the modelview matrix when `gl:light` is called, and it is stored in eye coordinates. It is significant only when `?GL_SPOT_CUTOFF` is not 180, which it is initially. The initial direction is (0 0 -1).

`?GL_SPOT_EXPONENT`: `Params` is a single integer or floating-point value that specifies the intensity distribution of the light. Integer and floating-point values are mapped directly. Only values in the range [0 128] are accepted.

Effective light intensity is attenuated by the cosine of the angle between the direction of the light and the direction from the light to the vertex being lighted, raised to the power of the spot exponent. Thus, higher spot exponents result in a more focused light source, regardless of the spot cutoff angle (see `?GL_SPOT_CUTOFF`, next paragraph). The initial spot exponent is 0, resulting in uniform light distribution.

`?GL_SPOT_CUTOFF`: `Params` is a single integer or floating-point value that specifies the maximum spread angle of a light source. Integer and floating-point values are mapped directly. Only values in the range [0 90] and the special value 180 are accepted. If the angle between the direction of the light and the direction from the light to the vertex being lighted is greater than the spot cutoff angle, the light is completely masked. Otherwise, its intensity is controlled by the spot exponent and the attenuation factors. The initial spot cutoff is 180, resulting in uniform light distribution.

`?GL_CONSTANT_ATTENUATION`

`?GL_LINEAR_ATTENUATION`

`?GL_QUADRATIC_ATTENUATION`: `Params` is a single integer or floating-point value that specifies one of the three light attenuation factors. Integer and floating-point values are mapped directly. Only nonnegative values are accepted. If the light is positional, rather than directional, its intensity is attenuated by the reciprocal of the sum of the constant factor, the linear factor times the distance between the light and the vertex being lighted, and the quadratic factor times the square of the same distance. The initial attenuation factors are (1, 0, 0), resulting in no attenuation.

See **external** documentation.

`lighti(Light, Pname, Param) -> ok`

Types:

```
Light = enum()
Pname = enum()
Param = integer()
```

See *lightf/3*

lightfv(Light, Pname, Params) -> ok

Types:

Light = enum()

Pname = enum()

Params = {float()}

See *lightf/3*

lightiv(Light, Pname, Params) -> ok

Types:

Light = enum()

Pname = enum()

Params = {integer()}

See *lightf/3*

getLightfv(Light, Pname) -> {float(), float(), float(), float()}

Types:

Light = enum()

Pname = enum()

Return light source parameter values

`gl:getLight` returns in `Params` the value or values of a light source parameter. `Light` names the light and is a symbolic name of the form `?GL_LIGHT` i where i ranges from 0 to the value of `?GL_MAX_LIGHTS` - 1. `?GL_MAX_LIGHTS` is an implementation dependent constant that is greater than or equal to eight. `Pname` specifies one of ten light source parameters, again by symbolic name.

The following parameters are defined:

`?GL_AMBIENT`: `Params` returns four integer or floating-point values representing the ambient intensity of the light source. Integer values, when requested, are linearly mapped from the internal floating-point representation such that 1.0 maps to the most positive representable integer value, and -1.0 maps to the most negative representable integer value. If the internal value is outside the range [-1 1], the corresponding integer return value is undefined. The initial value is (0, 0, 0, 1).

`?GL_DIFFUSE`: `Params` returns four integer or floating-point values representing the diffuse intensity of the light source. Integer values, when requested, are linearly mapped from the internal floating-point representation such that 1.0 maps to the most positive representable integer value, and -1.0 maps to the most negative representable integer value. If the internal value is outside the range [-1 1], the corresponding integer return value is undefined. The initial value for `?GL_LIGHT0` is (1, 1, 1, 1); for other lights, the initial value is (0, 0, 0, 0).

`?GL_SPECULAR`: `Params` returns four integer or floating-point values representing the specular intensity of the light source. Integer values, when requested, are linearly mapped from the internal floating-point representation such that 1.0 maps to the most positive representable integer value, and -1.0 maps to the most negative representable integer value. If the internal value is outside the range [-1 1], the corresponding integer return value is undefined. The initial value for `?GL_LIGHT0` is (1, 1, 1, 1); for other lights, the initial value is (0, 0, 0, 0).

`?GL_POSITION`: `Params` returns four integer or floating-point values representing the position of the light source. Integer values, when requested, are computed by rounding the internal floating-point values to the nearest integer value. The returned values are those maintained in eye coordinates. They will not be equal to the values specified using *gl:lightf/3* , unless the modelview matrix was identity at the time *gl:lightf/3* was called. The initial value is (0, 0, 1, 0).

`?GL_SPOT_DIRECTION`: `Params` returns three integer or floating-point values representing the direction of the light source. Integer values, when requested, are computed by rounding the internal floating-point values to the nearest

integer value. The returned values are those maintained in eye coordinates. They will not be equal to the values specified using *gl:lightf/3* , unless the modelview matrix was identity at the time *gl:lightf/3* was called. Although spot direction is normalized before being used in the lighting equation, the returned values are the transformed versions of the specified values prior to normalization. The initial value is (0 0 -1).

?GL_SPOT_EXPONENT: Params returns a single integer or floating-point value representing the spot exponent of the light. An integer value, when requested, is computed by rounding the internal floating-point representation to the nearest integer. The initial value is 0.

?GL_SPOT_CUTOFF: Params returns a single integer or floating-point value representing the spot cutoff angle of the light. An integer value, when requested, is computed by rounding the internal floating-point representation to the nearest integer. The initial value is 180.

?GL_CONSTANT_ATTENUATION: Params returns a single integer or floating-point value representing the constant (not distance-related) attenuation of the light. An integer value, when requested, is computed by rounding the internal floating-point representation to the nearest integer. The initial value is 1.

?GL_LINEAR_ATTENUATION: Params returns a single integer or floating-point value representing the linear attenuation of the light. An integer value, when requested, is computed by rounding the internal floating-point representation to the nearest integer. The initial value is 0.

?GL_QUADRATIC_ATTENUATION: Params returns a single integer or floating-point value representing the quadratic attenuation of the light. An integer value, when requested, is computed by rounding the internal floating-point representation to the nearest integer. The initial value is 0.

See **external** documentation.

getLightiv(Light, Pname) -> {integer(), integer(), integer(), integer()}

Types:

```
Light = enum()
Pname = enum()
```

See *getLightfv/2*

lightModelf(Pname, Param) -> ok

Types:

```
Pname = enum()
Param = float()
```

Set the lighting model parameters

gl:lightModel sets the lighting model parameter. Pname names a parameter and Params gives the new value. There are three lighting model parameters:

?GL_LIGHT_MODEL_AMBIENT: Params contains four integer or floating-point values that specify the ambient RGBA intensity of the entire scene. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient scene intensity is (0.2, 0.2, 0.2, 1.0).

?GL_LIGHT_MODEL_COLOR_CONTROL: Params must be either ?GL_SEPARATE_SPECULAR_COLOR or ?GL_SINGLE_COLOR. ?GL_SINGLE_COLOR specifies that a single color is generated from the lighting computation for a vertex. ?GL_SEPARATE_SPECULAR_COLOR specifies that the specular color computation of lighting be stored separately from the remainder of the lighting computation. The specular color is summed into the generated fragment's color after the application of texture mapping (if enabled). The initial value is ?GL_SINGLE_COLOR.

?GL_LIGHT_MODEL_LOCAL_VIEWER: Params is a single integer or floating-point value that specifies how specular reflection angles are computed. If Params is 0 (or 0.0), specular reflection angles take the view direction to

be parallel to and in the direction of the -z axis, regardless of the location of the vertex in eye coordinates. Otherwise, specular reflections are computed from the origin of the eye coordinate system. The initial value is 0.

?GL_LIGHT_MODEL_TWO_SIDE: Params is a single integer or floating-point value that specifies whether one- or two-sided lighting calculations are done for polygons. It has no effect on the lighting calculations for points, lines, or bitmaps. If Params is 0 (or 0.0), one-sided lighting is specified, and only the front material parameters are used in the lighting equation. Otherwise, two-sided lighting is specified. In this case, vertices of back-facing polygons are lighted using the back material parameters and have their normals reversed before the lighting equation is evaluated. Vertices of front-facing polygons are always lighted using the front material parameters, with no change to their normals. The initial value is 0.

In RGBA mode, the lighted color of a vertex is the sum of the material emission intensity, the product of the material ambient reflectance and the lighting model full-scene ambient intensity, and the contribution of each enabled light source. Each light source contributes the sum of three terms: ambient, diffuse, and specular. The ambient light source contribution is the product of the material ambient reflectance and the light's ambient intensity. The diffuse light source contribution is the product of the material diffuse reflectance, the light's diffuse intensity, and the dot product of the vertex's normal with the normalized vector from the vertex to the light source. The specular light source contribution is the product of the material specular reflectance, the light's specular intensity, and the dot product of the normalized vertex-to-eye and vertex-to-light vectors, raised to the power of the shininess of the material. All three light source contributions are attenuated equally based on the distance from the vertex to the light source and on light source direction, spread exponent, and spread cutoff angle. All dot products are replaced with 0 if they evaluate to a negative value.

The alpha component of the resulting lighted color is set to the alpha value of the material diffuse reflectance.

In color index mode, the value of the lighted index of a vertex ranges from the ambient to the specular values passed to *gl:materialf/3* using ?GL_COLOR_INDEXES. Diffuse and specular coefficients, computed with a (.30, .59, .11) weighting of the lights' colors, the shininess of the material, and the same reflection and attenuation equations as in the RGBA case, determine how much above ambient the resulting index is.

See **external** documentation.

lightModeli(Pname, Param) -> ok

Types:

Pname = enum()

Param = integer()

See *lightModelf/2*

lightModelfv(Pname, Params) -> ok

Types:

Pname = enum()

Params = {float()}

See *lightModelf/2*

lightModeliv(Pname, Params) -> ok

Types:

Pname = enum()

Params = {integer()}

See *lightModelf/2*

materialf(Face, Pname, Param) -> ok

Types:

Face = enum()

Pname = enum()

Param = float()

Specify material parameters for the lighting model

`gl:material` assigns values to material parameters. There are two matched sets of material parameters. One, the `front-facing` set, is used to shade points, lines, bitmaps, and all polygons (when two-sided lighting is disabled), or just front-facing polygons (when two-sided lighting is enabled). The other set, `back-facing`, is used to shade back-facing polygons only when two-sided lighting is enabled. Refer to the *gl:lightModelf/2* reference page for details concerning one- and two-sided lighting calculations.

`gl:material` takes three arguments. The first, `Face` , specifies whether the `?GL_FRONT` materials, the `?GL_BACK` materials, or both `?GL_FRONT_AND_BACK` materials will be modified. The second, `Pname` , specifies which of several parameters in one or both sets will be modified. The third, `Params` , specifies what value or values will be assigned to the specified parameter.

Material parameters are used in the lighting equation that is optionally applied to each vertex. The equation is discussed in the *gl:lightModelf/2* reference page. The parameters that can be specified using `gl:material`, and their interpretations by the lighting equation, are as follows:

`?GL_AMBIENT`: `Params` contains four integer or floating-point values that specify the ambient RGBA reflectance of the material. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient reflectance for both front- and back-facing materials is (0.2, 0.2, 0.2, 1.0).

`?GL_DIFFUSE`: `Params` contains four integer or floating-point values that specify the diffuse RGBA reflectance of the material. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial diffuse reflectance for both front- and back-facing materials is (0.8, 0.8, 0.8, 1.0).

`?GL_SPECULAR`: `Params` contains four integer or floating-point values that specify the specular RGBA reflectance of the material. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial specular reflectance for both front- and back-facing materials is (0, 0, 0, 1).

`?GL_EMISSION`: `Params` contains four integer or floating-point values that specify the RGBA emitted light intensity of the material. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial emission intensity for both front- and back-facing materials is (0, 0, 0, 1).

`?GL_SHININESS`: `Params` is a single integer or floating-point value that specifies the RGBA specular exponent of the material. Integer and floating-point values are mapped directly. Only values in the range [0 128] are accepted. The initial specular exponent for both front- and back-facing materials is 0.

`?GL_AMBIENT_AND_DIFFUSE`: Equivalent to calling `gl:material` twice with the same parameter values, once with `?GL_AMBIENT` and once with `?GL_DIFFUSE`.

`?GL_COLOR_INDEXES`: `Params` contains three integer or floating-point values specifying the color indices for ambient, diffuse, and specular lighting. These three values, and `?GL_SHININESS`, are the only material values used by the color index mode lighting equation. Refer to the *gl:lightModelf/2* reference page for a discussion of color index lighting.

See **external** documentation.

materiali(Face, Pname, Param) -> ok

Types:

Face = enum()
Pname = enum()
Param = integer()

See *materialf/3*

materialfv(Face, Pname, Params) -> ok

Types:

Face = enum()
Pname = enum()
Params = {float()}

See *materialf/3*

materialiv(Face, Pname, Params) -> ok

Types:

Face = enum()
Pname = enum()
Params = {integer()}

See *materialf/3*

getMaterialfv(Face, Pname) -> {float(), float(), float(), float()}

Types:

Face = enum()
Pname = enum()

Return material parameters

`gl:getMaterial` returns in `Params` the value or values of parameter `Pname` of material `Face` . Six parameters are defined:

`?GL_AMBIENT`: `Params` returns four integer or floating-point values representing the ambient reflectance of the material. Integer values, when requested, are linearly mapped from the internal floating-point representation such that 1.0 maps to the most positive representable integer value, and -1.0 maps to the most negative representable integer value. If the internal value is outside the range [-1 1], the corresponding integer return value is undefined. The initial value is (0.2, 0.2, 0.2, 1.0)

`?GL_DIFFUSE`: `Params` returns four integer or floating-point values representing the diffuse reflectance of the material. Integer values, when requested, are linearly mapped from the internal floating-point representation such that 1.0 maps to the most positive representable integer value, and -1.0 maps to the most negative representable integer value. If the internal value is outside the range [-1 1], the corresponding integer return value is undefined. The initial value is (0.8, 0.8, 0.8, 1.0).

`?GL_SPECULAR`: `Params` returns four integer or floating-point values representing the specular reflectance of the material. Integer values, when requested, are linearly mapped from the internal floating-point representation such that 1.0 maps to the most positive representable integer value, and -1.0 maps to the most negative representable integer value. If the internal value is outside the range [-1 1], the corresponding integer return value is undefined. The initial value is (0, 0, 0, 1).

?GL_EMISSION: Params returns four integer or floating-point values representing the emitted light intensity of the material. Integer values, when requested, are linearly mapped from the internal floating-point representation such that 1.0 maps to the most positive representable integer value, and -1.0 maps to the most negative representable integer value. If the internal value is outside the range [-1 1], the corresponding integer return value is undefined. The initial value is (0, 0, 0, 1).

?GL_SHININESS: Params returns one integer or floating-point value representing the specular exponent of the material. Integer values, when requested, are computed by rounding the internal floating-point value to the nearest integer value. The initial value is 0.

?GL_COLOR_INDEXES: Params returns three integer or floating-point values representing the ambient, diffuse, and specular indices of the material. These indices are used only for color index lighting. (All the other parameters are used only for RGBA lighting.) Integer values, when requested, are computed by rounding the internal floating-point values to the nearest integer values.

See **external** documentation.

getMaterialiv(Face, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Face = enum()

Pname = enum()

See *getMaterialfv/2*

colorMaterial(Face, Mode) -> ok

Types:

Face = enum()

Mode = enum()

Cause a material color to track the current color

gl:colorMaterial specifies which material parameters track the current color. When ?GL_COLOR_MATERIAL is enabled, the material parameter or parameters specified by Mode , of the material or materials specified by Face , track the current color at all times.

To enable and disable ?GL_COLOR_MATERIAL, call *gl:enable/1* and *gl:enable/1* with argument ?GL_COLOR_MATERIAL. ?GL_COLOR_MATERIAL is initially disabled.

See **external** documentation.

pixelZoom(Xfactor, Yfactor) -> ok

Types:

Xfactor = float()

Yfactor = float()

Specify the pixel zoom factors

gl:pixelZoom specifies values for the x and y zoom factors. During the execution of *gl:drawPixels/5* or *gl:copyPixels/5* , if (xr, yr) is the current raster position, and a given element is in the mth row and nth column of the pixel rectangle, then pixels whose centers are in the rectangle with corners at

(xr+n. xfactor, yr+m. yfactor)

(xr+(n+1). xfactor, yr+(m+1). yfactor)

are candidates for replacement. Any pixel whose center lies on the bottom or left edge of this rectangular region is also modified.

Pixel zoom factors are not limited to positive values. Negative zoom factors reflect the resulting image about the current raster position.

See **external** documentation.

pixelStoref(Pname, Param) -> ok

Types:

Pname = enum()

Param = float()

Set pixel storage modes

`gl:pixelStore` sets pixel storage modes that affect the operation of subsequent *gl:readPixels/7* as well as the unpacking of texture patterns (see *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7*), *gl:compressedTexImage1D/7* , *gl:compressedTexImage2D/8* , *gl:compressedTexImage3D/9* , *gl:compressedTexSubImage1D/7* , *gl:compressedTexSubImage2D/9* or *gl:compressedTexSubImage1D/7* .

`Pname` is a symbolic constant indicating the parameter to be set, and `Param` is the new value. Six of the twelve storage parameters affect how pixel data is returned to client memory. They are as follows:

`?GL_PACK_SWAP_BYTES`: If true, byte ordering for multibyte color components, depth components, or stencil indices is reversed. That is, if a four-byte component consists of bytes b 0, b 1, b 2, b 3, it is stored in memory as b 3, b 2, b 1, b 0 if `?GL_PACK_SWAP_BYTES` is true. `?GL_PACK_SWAP_BYTES` has no effect on the memory order of components within a pixel, only on the order of bytes within components or indices. For example, the three components of a `?GL_RGB` format pixel are always stored with red first, green second, and blue third, regardless of the value of `?GL_PACK_SWAP_BYTES`.

`?GL_PACK_LSB_FIRST`: If true, bits are ordered within a byte from least significant to most significant; otherwise, the first bit in each byte is the most significant one.

`?GL_PACK_ROW_LENGTH`: If greater than 0, `?GL_PACK_ROW_LENGTH` defines the number of pixels in a row. If the first pixel of a row is placed at location p in memory, then the location of the first pixel of the next row is obtained by skipping

k={n l(a/s) |(s n l)/a| s>= a s< a)

components or indices, where n is the number of components or indices in a pixel, l is the number of pixels in a row (`?GL_PACK_ROW_LENGTH` if it is greater than 0, the width argument to the pixel routine otherwise), a is the value of `?GL_PACK_ALIGNMENT` , and s is the size, in bytes, of a single component (if a< s, then it is as if a= s). In the case of 1-bit values, the location of the next row is obtained by skipping

k= 8 a |(n l)/(8 a)|

components or indices.

The word `component` in this description refers to the nonindex values red, green, blue, alpha, and depth. Storage format `?GL_RGB`, for example, has three components per pixel: first red, then green, and finally blue.

`?GL_PACK_IMAGE_HEIGHT`: If greater than 0, `?GL_PACK_IMAGE_HEIGHT` defines the number of pixels in an image three-dimensional texture volume, where `image` is defined by all pixels sharing the same third dimension index. If the first pixel of a row is placed at location p in memory, then the location of the first pixel of the next row is obtained by skipping

k={n l h(a/s) |(s n l h)/a| s>= a s< a)

components or indices, where n is the number of components or indices in a pixel, l is the number of pixels in a row (`?GL_PACK_ROW_LENGTH` if it is greater than 0, the width argument to *gl:texImage3D/10* otherwise), h is the number of rows in a pixel image (`?GL_PACK_IMAGE_HEIGHT` if it is greater than 0, the height argument to the

gl:texImage3D/10 routine otherwise), a is the value of ?GL_PACK_ALIGNMENT , and s is the size, in bytes, of a single component (if a< s, then it is as if a= s).

The word component in this description refers to the nonindex values red, green, blue, alpha, and depth. Storage format ?GL_RGB, for example, has three components per pixel: first red, then green, and finally blue.

?GL_PACK_SKIP_PIXELS, ?GL_PACK_SKIP_ROWS, and ?GL_PACK_SKIP_IMAGES

These values are provided as a convenience to the programmer; they provide no functionality that cannot be duplicated simply by incrementing the pointer passed to *gl:readPixels/7* . Setting ?GL_PACK_SKIP_PIXELS to i is equivalent to incrementing the pointer by i n components or indices, where n is the number of components or indices in each pixel. Setting ?GL_PACK_SKIP_ROWS to j is equivalent to incrementing the pointer by j m components or indices, where m is the number of components or indices per row, as just computed in the ?GL_PACK_ROW_LENGTH section. Setting ?GL_PACK_SKIP_IMAGES to k is equivalent to incrementing the pointer by k p, where p is the number of components or indices per image, as computed in the ?GL_PACK_IMAGE_HEIGHT section.

?GL_PACK_ALIGNMENT: Specifies the alignment requirements for the start of each pixel row in memory. The allowable values are 1 (byte-alignment), 2 (rows aligned to even-numbered bytes), 4 (word-alignment), and 8 (rows start on double-word boundaries).

The other six of the twelve storage parameters affect how pixel data is read from client memory. These values are significant for *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7* , and *gl:texSubImage1D/7*

They are as follows:

?GL_UNPACK_SWAP_BYTES: If true, byte ordering for multibyte color components, depth components, or stencil indices is reversed. That is, if a four-byte component consists of bytes b 0, b 1, b 2, b 3, it is taken from memory as b 3, b 2, b 1, b 0 if ?GL_UNPACK_SWAP_BYTES is true. ?GL_UNPACK_SWAP_BYTES has no effect on the memory order of components within a pixel, only on the order of bytes within components or indices. For example, the three components of a ?GL_RGB format pixel are always stored with red first, green second, and blue third, regardless of the value of ?GL_UNPACK_SWAP_BYTES.

?GL_UNPACK_LSB_FIRST: If true, bits are ordered within a byte from least significant to most significant; otherwise, the first bit in each byte is the most significant one.

?GL_UNPACK_ROW_LENGTH: If greater than 0, ?GL_UNPACK_ROW_LENGTH defines the number of pixels in a row. If the first pixel of a row is placed at location p in memory, then the location of the first pixel of the next row is obtained by skipping

k={n l(a/s) |(s n l)/a| s>= a s< a)

components or indices, where n is the number of components or indices in a pixel, l is the number of pixels in a row (?GL_UNPACK_ROW_LENGTH if it is greater than 0, the width argument to the pixel routine otherwise), a is the value of ?GL_UNPACK_ALIGNMENT , and s is the size, in bytes, of a single component (if a< s, then it is as if a= s). In the case of 1-bit values, the location of the next row is obtained by skipping

k= 8 a |(n l)/(8 a)|

components or indices.

The word component in this description refers to the nonindex values red, green, blue, alpha, and depth. Storage format ?GL_RGB, for example, has three components per pixel: first red, then green, and finally blue.

?GL_UNPACK_IMAGE_HEIGHT: If greater than 0, ?GL_UNPACK_IMAGE_HEIGHT defines the number of pixels in an image of a three-dimensional texture volume. Where image is defined by all pixel sharing the same third dimension index. If the first pixel of a row is placed at location p in memory, then the location of the first pixel of the next row is obtained by skipping

k={n l h(a/s) |(s n l h)/a| s>= a s< a)

components or indices, where n is the number of components or indices in a pixel, l is the number of pixels in a row (?GL_UNPACK_ROW_LENGTH if it is greater than 0, the width argument to *gl:texImage3D/10* otherwise), h is the number of rows in an image (?GL_UNPACK_IMAGE_HEIGHT if it is greater than 0, the height argument to *gl:texImage3D/10* otherwise), a is the value of ?GL_UNPACK_ALIGNMENT, and s is the size, in bytes, of a single component (if a< s, then it is as if a= s).

The word component in this description refers to the nonindex values red, green, blue, alpha, and depth. Storage format ?GL_RGB, for example, has three components per pixel: first red, then green, and finally blue.

?GL_UNPACK_SKIP_PIXELS and ?GL_UNPACK_SKIP_ROWS

These values are provided as a convenience to the programmer; they provide no functionality that cannot be duplicated by incrementing the pointer passed to *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texSubImage1D/7* or *gl:texSubImage1D/7* . Setting ?GL_UNPACK_SKIP_PIXELS to i is equivalent to incrementing the pointer by i n components or indices, where n is the number of components or indices in each pixel. Setting ?GL_UNPACK_SKIP_ROWS to j is equivalent to incrementing the pointer by j k components or indices, where k is the number of components or indices per row, as just computed in the ?GL_UNPACK_ROW_LENGTH section.

?GL_UNPACK_ALIGNMENT: Specifies the alignment requirements for the start of each pixel row in memory. The allowable values are 1 (byte-alignment), 2 (rows aligned to even-numbered bytes), 4 (word-alignment), and 8 (rows start on double-word boundaries).

The following table gives the type, initial value, and range of valid values for each storage parameter that can be set with gl:pixelStore.

Pname	Type	Initial Value	Valid Range
?GL_PACK_SWAP_BYTES	boolean	false	true or false
?GL_PACK_LSB_FIRST	boolean	false	true or false
?GL_PACK_ROW_LENGTH	integer	0	[0)
?GL_PACK_IMAGE_HEIGHT	integer	0	[0)
?GL_PACK_SKIP_ROWS	integer	0	[0)
?GL_PACK_SKIP_PIXELS	integer	0	[0)
?GL_PACK_SKIP_IMAGES	integer	0	[0)
?GL_PACK_ALIGNMENT	integer	4	1, 2, 4, or 8
?GL_UNPACK_SWAP_BYTES	boolean	false	true or false
?GL_UNPACK_LSB_FIRST	boolean	false	true or false
?GL_UNPACK_ROW_LENGTH	integer	0	[0)
?GL_UNPACK_IMAGE_HEIGHT	integer	0	[0)
?GL_UNPACK_SKIP_ROWS	integer	0	[0)
?GL_UNPACK_SKIP_PIXELS	integer	0	[0)
?GL_UNPACK_SKIP_IMAGES	integer	0	[0)
?GL_UNPACK_ALIGNMENT	integer	4	1, 2, 4, or 8

gl:pixelStoref can be used to set any pixel store parameter. If the parameter type is boolean, then if Param is 0, the parameter is false; otherwise it is set to true. If Pname is a integer type parameter, Param is rounded to the nearest integer.

Likewise, gl:pixelStorei can also be used to set any of the pixel store parameters. Boolean parameters are set to false if Param is 0 and true otherwise.

See **external** documentation.

pixelStorei(Pname, Param) -> ok

Types:

Pname = enum()

Param = integer()

See *pixelStoref/2*

pixelTransferf(Pname, Param) -> ok

Types:

Pname = enum()

Param = float()

Set pixel transfer modes

`gl:pixelTransfer` sets pixel transfer modes that affect the operation of subsequent *gl:copyPixels/5* , *gl:copyTexImage1D/7* , *gl:copyTexImage2D/8* , *gl:copyTexSubImage1D/6* , *gl:copyTexSubImage2D/8* , *gl:copyTexSubImage3D/9* , *gl:drawPixels/5* , *gl:readPixels/7* , *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7* , and *gl:texSubImage1D/7* commands. Additionally, if the ARB_imaging subset is supported, the routines *gl:colorTable/6* , *gl:colorSubTable/6* , *gl:convolutionFilter1D/6* , *gl:convolutionFilter2D/7* , *gl:histogram/4* , *gl:minmax/3* , and *gl:separableFilter2D/8* are also affected. The algorithms that are specified by pixel transfer modes operate on pixels after they are read from the frame buffer (*gl:copyPixels/5* *gl:copyTexImage1D/7* , *gl:copyTexImage2D/8* , *gl:copyTexSubImage1D/6* , *gl:copyTexSubImage2D/8* , *gl:copyTexSubImage3D/9* , and *gl:readPixels/7*), or unpacked from client memory (*gl:drawPixels/5* , *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7* , and *gl:texSubImage1D/7*). Pixel transfer operations happen in the same order, and in the same manner, regardless of the command that resulted in the pixel operation. Pixel storage modes (see *gl:pixelStoref/2*) control the unpacking of pixels being read from client memory and the packing of pixels being written back into client memory.

Pixel transfer operations handle four fundamental pixel types: `color`, `color index` , `depth`, and `stencil`. `Color` pixels consist of four floating-point values with unspecified mantissa and exponent sizes, scaled such that 0 represents zero intensity and 1 represents full intensity. `Color indices` comprise a single fixed-point value, with unspecified precision to the right of the binary point. `Depth` pixels comprise a single floating-point value, with unspecified mantissa and exponent sizes, scaled such that 0.0 represents the minimum depth buffer value, and 1.0 represents the maximum depth buffer value. Finally, `stencil` pixels comprise a single fixed-point value, with unspecified precision to the right of the binary point.

The pixel transfer operations performed on the four basic pixel types are as follows:

`Color`: Each of the four color components is multiplied by a scale factor, then added to a bias factor. That is, the red component is multiplied by `?GL_RED_SCALE`, then added to `?GL_RED_BIAS`; the green component is multiplied by `?GL_GREEN_SCALE` , then added to `?GL_GREEN_BIAS`; the blue component is multiplied by `?GL_BLUE_SCALE` , then added to `?GL_BLUE_BIAS`; and the alpha component is multiplied by `?GL_ALPHA_SCALE` , then added to `?GL_ALPHA_BIAS`. After all four color components are scaled and biased, each is clamped to the range [0 1]. All color, scale, and bias values are specified with `gl:pixelTransfer`.

If `?GL_MAP_COLOR` is true, each color component is scaled by the size of the corresponding color-to-color map, then replaced by the contents of that map indexed by the scaled component. That is, the red component is scaled by `?GL_PIXEL_MAP_R_TO_R_SIZE`, then replaced by the contents of `?GL_PIXEL_MAP_R_TO_R` indexed by itself. The green component is scaled by `?GL_PIXEL_MAP_G_TO_G_SIZE`, then replaced by the contents of `?GL_PIXEL_MAP_G_TO_G` indexed by itself. The blue component is scaled by `?GL_PIXEL_MAP_B_TO_B_SIZE`, then replaced by the contents of `?GL_PIXEL_MAP_B_TO_B` indexed by itself. And the alpha component is scaled by `?GL_PIXEL_MAP_A_TO_A_SIZE`, then replaced by the contents of `?GL_PIXEL_MAP_A_TO_A` indexed by itself. All components taken from the maps are then clamped to the range [0 1]. `?GL_MAP_COLOR` is specified with `gl:pixelTransfer`. The contents of the various maps are specified with *gl:pixelMapfv/3* .

If the ARB_imaging extension is supported, each of the four color components may be scaled and biased after transformation by the color matrix. That is, the red component is multiplied by `?GL_POST_COLOR_MATRIX_RED_SCALE`, then added to `?GL_POST_COLOR_MATRIX_RED_BIAS` ; the green component is multiplied by `?GL_POST_COLOR_MATRIX_GREEN_SCALE`, then added to `?GL_POST_COLOR_MATRIX_GREEN_BIAS`; the blue component is multiplied by `?`

GL_POST_COLOR_MATRIX_BLUE_SCALE , then added to ?GL_POST_COLOR_MATRIX_BLUE_BIAS; and the alpha component is multiplied by ?GL_POST_COLOR_MATRIX_ALPHA_SCALE, then added to ?GL_POST_COLOR_MATRIX_ALPHA_BIAS . After all four color components are scaled and biased, each is clamped to the range [0 1].

Similarly, if the ARB_imaging extension is supported, each of the four color components may be scaled and biased after processing by the enabled convolution filter. That is, the red component is multiplied by ?GL_POST_CONVOLUTION_RED_SCALE, then added to ?GL_POST_CONVOLUTION_RED_BIAS ; the green component is multiplied by ?GL_POST_CONVOLUTION_GREEN_SCALE, then added to ?GL_POST_CONVOLUTION_GREEN_BIAS; the blue component is multiplied by ?GL_POST_CONVOLUTION_BLUE_SCALE , then added to ?GL_POST_CONVOLUTION_BLUE_BIAS; and the alpha component is multiplied by ?GL_POST_CONVOLUTION_ALPHA_SCALE, then added to ?GL_POST_CONVOLUTION_ALPHA_BIAS . After all four color components are scaled and biased, each is clamped to the range [0 1].

Color index: Each color index is shifted left by ?GL_INDEX_SHIFT bits; any bits beyond the number of fraction bits carried by the fixed-point index are filled with zeros. If ?GL_INDEX_SHIFT is negative, the shift is to the right, again zero filled. Then ?GL_INDEX_OFFSET is added to the index. ?GL_INDEX_SHIFT and ?GL_INDEX_OFFSET are specified with gl:pixelTransfer.

From this point, operation diverges depending on the required format of the resulting pixels. If the resulting pixels are to be written to a color index buffer, or if they are being read back to client memory in ?GL_COLOR_INDEX format, the pixels continue to be treated as indices. If ?GL_MAP_COLOR is true, each index is masked by 2 n-1 , where n is ?GL_PIXEL_MAP_I_TO_I_SIZE, then replaced by the contents of ?GL_PIXEL_MAP_I_TO_I indexed by the masked value. ?GL_MAP_COLOR is specified with gl:pixelTransfer . The contents of the index map is specified with *gl:pixelMapfv/3* .

If the resulting pixels are to be written to an RGBA color buffer, or if they are read back to client memory in a format other than ?GL_COLOR_INDEX, the pixels are converted from indices to colors by referencing the four maps ?GL_PIXEL_MAP_I_TO_R, ?GL_PIXEL_MAP_I_TO_G , ?GL_PIXEL_MAP_I_TO_B, and ?GL_PIXEL_MAP_I_TO_A. Before being dereferenced, the index is masked by 2 n-1, where n is ?GL_PIXEL_MAP_I_TO_R_SIZE for the red map, ?GL_PIXEL_MAP_I_TO_G_SIZE for the green map, ?GL_PIXEL_MAP_I_TO_B_SIZE for the blue map, and ?GL_PIXEL_MAP_I_TO_A_SIZE for the alpha map. All components taken from the maps are then clamped to the range [0 1]. The contents of the four maps is specified with *gl:pixelMapfv/3* .

Depth: Each depth value is multiplied by ?GL_DEPTH_SCALE, added to ?GL_DEPTH_BIAS , then clamped to the range [0 1].

Stencil: Each index is shifted ?GL_INDEX_SHIFT bits just as a color index is, then added to ?GL_INDEX_OFFSET. If ?GL_MAP_STENCIL is true, each index is masked by 2 n-1, where n is ?GL_PIXEL_MAP_S_TO_S_SIZE, then replaced by the contents of ?GL_PIXEL_MAP_S_TO_S indexed by the masked value.

The following table gives the type, initial value, and range of valid values for each of the pixel transfer parameters that are set with gl:pixelTransfer.

Pname Type Initial Value Valid Range
?GL_MAP_COLOR boolean false true/false
?GL_MAP_STENCIL boolean false true/false
?GL_INDEX_SHIFT integer 0 (-)
?GL_INDEX_OFFSET integer 0 (-)
?GL_RED_SCALE float 1 (-)
?GL_GREEN_SCALE float 1 (-)
?GL_BLUE_SCALE float 1 (-)
?GL_ALPHA_SCALE float 1 (-)
?GL_DEPTH_SCALE float 1 (-)

?GL_RED_BIAS float 0 (-)
?GL_GREEN_BIAS float 0 (-)
?GL_BLUE_BIAS float 0 (-)
?GL_ALPHA_BIAS float 0 (-)
?GL_DEPTH_BIAS float 0 (-)
?GL_POST_COLOR_MATRIX_RED_SCALE float 1 (-)
?GL_POST_COLOR_MATRIX_GREEN_SCALE float 1 (-)
?GL_POST_COLOR_MATRIX_BLUE_SCALE float 1 (-)
?GL_POST_COLOR_MATRIX_ALPHA_SCALE float 1 (-)
?GL_POST_COLOR_MATRIX_RED_BIAS float 0 (-)
?GL_POST_COLOR_MATRIX_GREEN_BIAS float 0 (-)
?GL_POST_COLOR_MATRIX_BLUE_BIAS float 0 (-)
?GL_POST_COLOR_MATRIX_ALPHA_BIAS float 0 (-)
?GL_POST_CONVOLUTION_RED_SCALE float 1 (-)
?GL_POST_CONVOLUTION_GREEN_SCALE float 1 (-)
?GL_POST_CONVOLUTION_BLUE_SCALE float 1 (-)
?GL_POST_CONVOLUTION_ALPHA_SCALE float 1 (-)
?GL_POST_CONVOLUTION_RED_BIAS float 0 (-)
?GL_POST_CONVOLUTION_GREEN_BIAS float 0 (-)
?GL_POST_CONVOLUTION_BLUE_BIAS float 0 (-)
?GL_POST_CONVOLUTION_ALPHA_BIAS float 0 (-)

`gl:pixelTransferf` can be used to set any pixel transfer parameter. If the parameter type is boolean, 0 implies false and any other value implies true. If `Pname` is an integer parameter, `Param` is rounded to the nearest integer.

Likewise, `gl:pixelTransferi` can be used to set any of the pixel transfer parameters. Boolean parameters are set to false if `Param` is 0 and to true otherwise. `Param` is converted to floating point before being assigned to real-valued parameters.

See **external** documentation.

`pixelTransferi(Pname, Param) -> ok`

Types:

`Pname = enum()`
`Param = integer()`

See *pixelTransferf/2*

`pixelMapfv(Map, Mapsize, Values) -> ok`

Types:

`Map = enum()`
`Mapsize = integer()`
`Values = binary()`

Set up pixel transfer maps

`gl:pixelMap` sets up translation tables, or `maps`, used by *gl:copyPixels/5* , *gl:copyTexImage1D/7* , *gl:copyTexImage2D/8* , *gl:copyTexSubImage1D/6* , *gl:copyTexSubImage2D/8* , *gl:copyTexSubImage3D/9* , *gl:drawPixels/5* , *gl:readPixels/7* , *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7* , and *gl:texSubImage1D/7* . Additionally, if the ARB_imaging subset is supported, the routines *gl:colorTable/6* , *gl:colorSubTable/6* , *gl:convolutionFilter1D/6* , *gl:convolutionFilter2D/7* , *gl:histogram/4* , *gl:minmax/3* , and *gl:separableFilter2D/8* . Use of these maps is described completely in the *gl:pixelTransferf/2*

reference page, and partly in the reference pages for the pixel and texture image commands. Only the specification of the maps is described in this reference page.

`Map` is a symbolic map name, indicating one of ten maps to set. `Mapsize` specifies the number of entries in the map, and `Values` is a pointer to an array of `Mapsize` map values.

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a pixel transfer map is specified, `Values` is treated as a byte offset into the buffer object's data store.

The ten maps are as follows:

`?GL_PIXEL_MAP_I_TO_I`: Maps color indices to color indices.

`?GL_PIXEL_MAP_S_TO_S`: Maps stencil indices to stencil indices.

`?GL_PIXEL_MAP_I_TO_R`: Maps color indices to red components.

`?GL_PIXEL_MAP_I_TO_G`: Maps color indices to green components.

`?GL_PIXEL_MAP_I_TO_B`: Maps color indices to blue components.

`?GL_PIXEL_MAP_I_TO_A`: Maps color indices to alpha components.

`?GL_PIXEL_MAP_R_TO_R`: Maps red components to red components.

`?GL_PIXEL_MAP_G_TO_G`: Maps green components to green components.

`?GL_PIXEL_MAP_B_TO_B`: Maps blue components to blue components.

`?GL_PIXEL_MAP_A_TO_A`: Maps alpha components to alpha components.

The entries in a map can be specified as single-precision floating-point numbers, unsigned short integers, or unsigned int integers. Maps that store color component values (all but `?GL_PIXEL_MAP_I_TO_I` and `?GL_PIXEL_MAP_S_TO_S`) retain their values in floating-point format, with unspecified mantissa and exponent sizes. Floating-point values specified by `gl:pixelMapfv` are converted directly to the internal floating-point format of these maps, then clamped to the range [0,1]. Unsigned integer values specified by `gl:pixelMapusv` and `gl:pixelMapuiv` are converted linearly such that the largest representable integer maps to 1.0, and 0 maps to 0.0.

Maps that store indices, `?GL_PIXEL_MAP_I_TO_I` and `?GL_PIXEL_MAP_S_TO_S`, retain their values in fixed-point format, with an unspecified number of bits to the right of the binary point. Floating-point values specified by `gl:pixelMapfv` are converted directly to the internal fixed-point format of these maps. Unsigned integer values specified by `gl:pixelMapusv` and `gl:pixelMapuiv` specify integer values, with all 0's to the right of the binary point.

The following table shows the initial sizes and values for each of the maps. Maps that are indexed by either color or stencil indices must have `Mapsize` = 2 n for some n or the results are undefined. The maximum allowable size for each map depends on the implementation and can be determined by calling *gl:getBooleanv/1* with argument `?GL_MAX_PIXEL_MAP_TABLE` . The single maximum applies to all maps; it is at least 32.

`Map` `Lookup Index` `Lookup Value` `Initial Size` `Initial Value`
`?GL_PIXEL_MAP_I_TO_I` color index color index 1 0
`?GL_PIXEL_MAP_S_TO_S` stencil index stencil index 1 0
`?GL_PIXEL_MAP_I_TO_R` color index R 1 0
`?GL_PIXEL_MAP_I_TO_G` color index G 1 0
`?GL_PIXEL_MAP_I_TO_B` color index B 1 0
`?GL_PIXEL_MAP_I_TO_A` color index A 1 0
`?GL_PIXEL_MAP_R_TO_R` R R 1 0
`?GL_PIXEL_MAP_G_TO_G` G G 1 0
`?GL_PIXEL_MAP_B_TO_B` B B 1 0
`?GL_PIXEL_MAP_A_TO_A` A A 1 0

See **external** documentation.

pixelMapuiv(Map, Mapsize, Values) -> ok

Types:

```
Map = enum()
Mapsize = integer()
Values = binary()
```

See *pixelMapfv/3*

pixelMapusv(Map, Mapsize, Values) -> ok

Types:

```
Map = enum()
Mapsize = integer()
Values = binary()
```

See *pixelMapfv/3*

getPixelMapfv(Map, Values) -> ok

Types:

```
Map = enum()
Values = mem()
```

Return the specified pixel map

See the *gl:pixelMapfv/3* reference page for a description of the acceptable values for the `Map` parameter. `gl:getPixelMap` returns in `Data` the contents of the pixel map specified in `Map` . Pixel maps are used during the execution of *gl:readPixels/7* , *gl:drawPixels/5* , *gl:copyPixels/5* , *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7* , *gl:copyTexImage1D/7* , *gl:copyTexImage2D/8* , *gl:copyTexSubImage1D/6* , *gl:copyTexSubImage2D/8* , and *gl:copyTexSubImage3D/9* . to map color indices, stencil indices, color components, and depth components to other values.

If a non-zero named buffer object is bound to the `?GL_PIXEL_PACK_BUFFER` target (see *gl:bindBuffer/2*) while a pixel map is requested, `Data` is treated as a byte offset into the buffer object's data store.

Unsigned integer values, if requested, are linearly mapped from the internal fixed or floating-point representation such that 1.0 maps to the largest representable integer value, and 0.0 maps to 0. Return unsigned integer values are undefined if the map value was not in the range [0,1].

To determine the required size of `Map` , call *gl:getBooleanv/1* with the appropriate symbolic constant.

See **external** documentation.

getPixelMapuiv(Map, Values) -> ok

Types:

```
Map = enum()
Values = mem()
```

See *getPixelMapfv/2*

getPixelMapusv(Map, Values) -> ok

Types:

```
Map = enum()
Values = mem()
```

See *getPixelMapfv/2*

bitmap(Width, Height, Xorig, Yorig, Xmove, Ymove, Bitmap) -> ok

Types:

Width = integer()
Height = integer()
Xorig = float()
Yorig = float()
Xmove = float()
Ymove = float()
Bitmap = offset() | mem()

Draw a bitmap

A bitmap is a binary image. When drawn, the bitmap is positioned relative to the current raster position, and frame buffer pixels corresponding to 1's in the bitmap are written using the current raster color or index. Frame buffer pixels corresponding to 0's in the bitmap are not modified.

`gl:bitmap` takes seven arguments. The first pair specifies the width and height of the bitmap image. The second pair specifies the location of the bitmap origin relative to the lower left corner of the bitmap image. The third pair of arguments specifies `x` and `y` offsets to be added to the current raster position after the bitmap has been drawn. The final argument is a pointer to the bitmap image itself.

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a bitmap image is specified, `Bitmap` is treated as a byte offset into the buffer object's data store.

The bitmap image is interpreted like image data for the *gl:drawPixels/5* command, with `Width` and `Height` corresponding to the width and height arguments of that command, and with `type` set to `?GL_BITMAP` and `format` set to `?GL_COLOR_INDEX` . Modes specified using *gl:pixelStoref/2* affect the interpretation of bitmap image data; modes specified using *gl:pixelTransferf/2* do not.

If the current raster position is invalid, `gl:bitmap` is ignored. Otherwise, the lower left corner of the bitmap image is positioned at the window coordinates

x w=|x r-x o|

y w=|y r-y o|

where (x r y r) is the raster position and (x o y o) is the bitmap origin. Fragments are then generated for each pixel corresponding to a 1 (one) in the bitmap image. These fragments are generated using the current raster `z` coordinate, color or color index, and current raster texture coordinates. They are then treated just as if they had been generated by a point, line, or polygon, including texture mapping, fogging, and all per-fragment operations such as alpha and depth testing.

After the bitmap has been drawn, the `x` and `y` coordinates of the current raster position are offset by `Xmove` and `Ymove` . No change is made to the `z` coordinate of the current raster position, or to the current raster color, texture coordinates, or index.

See **external** documentation.

readPixels(X, Y, Width, Height, Format, Type, Pixels) -> ok

Types:

X = integer()
Y = integer()
Width = integer()

```
Height = integer()
Format = enum()
Type = enum()
Pixels = mem()
```

Read a block of pixels from the frame buffer

`gl:readPixels` returns pixel data from the frame buffer, starting with the pixel whose lower left corner is at location (`X` , `Y`), into client memory starting at location `Data` . Several parameters control the processing of the pixel data before it is placed into client memory. These parameters are set with *gl:pixelStoref/2* . This reference page describes the effects on `gl:readPixels` of most, but not all of the parameters specified by these three commands.

If a non-zero named buffer object is bound to the `?GL_PIXEL_PACK_BUFFER` target (see *gl:bindBuffer/2*) while a block of pixels is requested, `Data` is treated as a byte offset into the buffer object's data store rather than a pointer to client memory.

`gl:readPixels` returns values from each pixel with lower left corner at (x+i y+j) for 0<= i< width and 0<= j< height. This pixel is said to be the ith pixel in the jth row. Pixels are returned in row order from the lowest to the highest row, left to right in each row.

`Format` specifies the format for the returned pixel values; accepted values are:

`?GL_STENCIL_INDEX`: Stencil values are read from the stencil buffer. Each index is converted to fixed point, shifted left or right depending on the value and sign of `?GL_INDEX_SHIFT` , and added to `?GL_INDEX_OFFSET`. If `?GL_MAP_STENCIL` is `?GL_TRUE`, indices are replaced by their mappings in the table `?GL_PIXEL_MAP_S_TO_S`.

`?GL_DEPTH_COMPONENT`: Depth values are read from the depth buffer. Each component is converted to floating point such that the minimum depth value maps to 0 and the maximum value maps to 1. Each component is then multiplied by `?GL_DEPTH_SCALE`, added to `?GL_DEPTH_BIAS` , and finally clamped to the range [0 1].

`?GL_DEPTH_STENCIL`: Values are taken from both the depth and stencil buffers. The `Type` parameter must be `?GL_UNSIGNED_INT_24_8` or `?GL_FLOAT_32_UNSIGNED_INT_24_8_REV` .

`?GL_RED`

`?GL_GREEN`

`?GL_BLUE`

`?GL_RGB`

`?GL_BGR`

`?GL_RGBA`

`?GL_BGRA`: Finally, the indices or components are converted to the proper format, as specified by `Type` . If `Format` is `?GL_STENCIL_INDEX` and `Type` is not `?GL_FLOAT`, each index is masked with the mask value given in the following table. If `Type` is `?GL_FLOAT`, then each integer index is converted to single-precision floating-point format.

If `Format` is `?GL_RED`, `?GL_GREEN`, `?GL_BLUE`, `?GL_RGB`, `?GL_BGR` , `?GL_RGBA`, or `?GL_BGRA` and `Type` is not `?GL_FLOAT`, each component is multiplied by the multiplier shown in the following table. If type is `?GL_FLOAT`, then each component is passed as is (or converted to the client's single-precision floating-point format if it is different from the one used by the GL).

`Type` `Index` `MaskComponent` `Conversion`
`?GL_UNSIGNED_BYTE` 2 8-1(2 8-1) c
`?GL_BYTE` 2 7-1((2 8-1) c-1)/2
`?GL_UNSIGNED_SHORT` 2 16-1(2 16-1) c
`?GL_SHORT` 2 15-1((2 16-1) c-1)/2
`?GL_UNSIGNED_INT` 2 32-1(2 32-1) c
`?GL_INT` 2 31-1((2 32-1) c-1)/2

?GL_HALF_FLOAT none c
?GL_FLOAT none c
?GL_UNSIGNED_BYTE_3_3_2 2 N-1(2 N-1) c
?GL_UNSIGNED_BYTE_2_3_3_REV 2 N-1(2 N-1) c
?GL_UNSIGNED_SHORT_5_6_5 2 N-1 (2 N-1) c
?GL_UNSIGNED_SHORT_5_6_5_REV 2 N-1(2 N-1) c
?GL_UNSIGNED_SHORT_4_4_4_4 2 N-1(2 N-1) c
?GL_UNSIGNED_SHORT_4_4_4_4_REV 2 N-1(2 N-1) c
?GL_UNSIGNED_SHORT_5_5_5_1 2 N-1(2 N-1) c
?GL_UNSIGNED_SHORT_1_5_5_5_REV 2 N-1 (2 N-1) c
?GL_UNSIGNED_INT_8_8_8_8 2 N-1(2 N-1) c
?GL_UNSIGNED_INT_8_8_8_8_REV 2 N-1(2 N-1) c
?GL_UNSIGNED_INT_10_10_10_2 2 N-1(2 N-1) c
?GL_UNSIGNED_INT_2_10_10_10_REV 2 N-1(2 N-1) c
?GL_UNSIGNED_INT_24_8 2 N-1(2 N-1) c
?GL_UNSIGNED_INT_10F_11F_11F_REV -- Special
?GL_UNSIGNED_INT_5_9_9_9_REV -- Special
?GL_FLOAT_32_UNSIGNED_INT_24_8_REV none c (Depth Only)

Return values are placed in memory as follows. If Format is ?GL_STENCIL_INDEX , ?GL_DEPTH_COMPONENT, ?GL_RED, ?GL_GREEN, or ?GL_BLUE, a single value is returned and the data for the ith pixel in the jth row is placed in location (j) width+i. ?GL_RGB and ?GL_BGR return three values, ?GL_RGBA and ?GL_BGRA return four values for each pixel, with all values corresponding to a single pixel occupying contiguous space in Data . Storage parameters set by *gl:pixelStoref/2* , such as ?GL_PACK_LSB_FIRST and ?GL_PACK_SWAP_BYTES, affect the way that data is written into memory. See *gl:pixelStoref/2* for a description.

See **external** documentation.

drawPixels(Width, Height, Format, Type, Pixels) -> ok

Types:

```
Width = integer()
Height = integer()
Format = enum()
Type = enum()
Pixels = offset() | mem()
```

Write a block of pixels to the frame buffer

gl:drawPixels reads pixel data from memory and writes it into the frame buffer relative to the current raster position, provided that the raster position is valid. Use *gl:rasterPos2d/2* or *gl:windowPos2d/2* to set the current raster position; use *gl:getBooleanv/1* with argument ?GL_CURRENT_RASTER_POSITION_VALID to determine if the specified raster position is valid, and *gl:getBooleanv/1* with argument ?GL_CURRENT_RASTER_POSITION to query the raster position.

Several parameters define the encoding of pixel data in memory and control the processing of the pixel data before it is placed in the frame buffer. These parameters are set with four commands: *gl:pixelStoref/2* , *gl:pixelTransferf/2* , *gl:pixelMapfv/3* , and *gl:pixelZoom/2* . This reference page describes the effects on gl:drawPixels of many, but not all, of the parameters specified by these four commands.

Data is read from Data as a sequence of signed or unsigned bytes, signed or unsigned shorts, signed or unsigned integers, or single-precision floating-point values, depending on Type . When Type is one of ?GL_UNSIGNED_BYTE, ?GL_BYTE, ?GL_UNSIGNED_SHORT , ?GL_SHORT, ?GL_UNSIGNED_INT, ?GL_INT, or ?GL_FLOAT each of these bytes, shorts, integers, or floating-point values is interpreted as one color or depth component, or one index, depending on Format . When Type is one of ?GL_UNSIGNED_BYTE_3_3_2 ,

?GL_UNSIGNED_SHORT_5_6_5, ?GL_UNSIGNED_SHORT_4_4_4_4, ?GL_UNSIGNED_SHORT_5_5_5_1 , ?GL_UNSIGNED_INT_8_8_8_8, or ?GL_UNSIGNED_INT_10_10_10_2, each unsigned value is interpreted as containing all the components for a single pixel, with the color components arranged according to Format . When Type is one of ?GL_UNSIGNED_BYTE_2_3_3_REV , ?GL_UNSIGNED_SHORT_5_6_5_REV, ?GL_UNSIGNED_SHORT_4_4_4_4_REV, ?GL_UNSIGNED_SHORT_1_5_5_5_REV , ?GL_UNSIGNED_INT_8_8_8_8_REV, or ?GL_UNSIGNED_INT_2_10_10_10_REV, each unsigned value is interpreted as containing all color components, specified by Format , for a single pixel in a reversed order. Indices are always treated individually. Color components are treated as groups of one, two, three, or four values, again based on Format . Both individual indices and groups of components are referred to as pixels. If Type is ?GL_BITMAP, the data must be unsigned bytes, and Format must be either ?GL_COLOR_INDEX or ?GL_STENCIL_INDEX. Each unsigned byte is treated as eight 1-bit pixels, with bit ordering determined by ?GL_UNPACK_LSB_FIRST (see *gl:pixelStoref/2*).

width*height pixels are read from memory, starting at location Data . By default, these pixels are taken from adjacent memory locations, except that after all Width pixels are read, the read pointer is advanced to the next four-byte boundary. The four-byte row alignment is specified by *gl:pixelStoref/2* with argument ?GL_UNPACK_ALIGNMENT , and it can be set to one, two, four, or eight bytes. Other pixel store parameters specify different read pointer advancements, both before the first pixel is read and after all Width pixels are read. See the *gl:pixelStoref/2* reference page for details on these options.

If a non-zero named buffer object is bound to the ?GL_PIXEL_UNPACK_BUFFER target (see *gl:bindBuffer/2*) while a block of pixels is specified, Data is treated as a byte offset into the buffer object's data store.

The width*height pixels that are read from memory are each operated on in the same way, based on the values of several parameters specified by *gl:pixelTransferf/2* and *gl:pixelMapfv/3* . The details of these operations, as well as the target buffer into which the pixels are drawn, are specific to the format of the pixels, as specified by Format . Format can assume one of 13 symbolic values:

?GL_COLOR_INDEX: Each pixel is a single value, a color index. It is converted to fixed-point format, with an unspecified number of bits to the right of the binary point, regardless of the memory data type. Floating-point values convert to true fixed-point values. Signed and unsigned integer data is converted with all fraction bits set to 0. Bitmap data convert to either 0 or 1.

Each fixed-point index is then shifted left by ?GL_INDEX_SHIFT bits and added to ?GL_INDEX_OFFSET . If ?GL_INDEX_SHIFT is negative, the shift is to the right. In either case, zero bits fill otherwise unspecified bit locations in the result.

If the GL is in RGBA mode, the resulting index is converted to an RGBA pixel with the help of the ?GL_PIXEL_MAP_I_TO_R, ?GL_PIXEL_MAP_I_TO_G, ?GL_PIXEL_MAP_I_TO_B , and ?GL_PIXEL_MAP_I_TO_A tables. If the GL is in color index mode, and if ?GL_MAP_COLOR is true, the index is replaced with the value that it references in lookup table ?GL_PIXEL_MAP_I_TO_I . Whether the lookup replacement of the index is done or not, the integer part of the index is then ANDed with 2 b-1, where b is the number of bits in a color index buffer.

The GL then converts the resulting indices or RGBA colors to fragments by attaching the current raster position z coordinate and texture coordinates to each pixel, then assigning x and y window coordinates to the nth fragment such that x n= x r+n% width

y n= y r+|n/width|

where (x r y r) is the current raster position. These pixel fragments are then treated just like the fragments generated by rasterizing points, lines, or polygons. Texture mapping, fog, and all the fragment operations are applied before the fragments are written to the frame buffer.

?GL_STENCIL_INDEX: Each pixel is a single value, a stencil index. It is converted to fixed-point format, with an unspecified number of bits to the right of the binary point, regardless of the memory data type. Floating-point values convert to true fixed-point values. Signed and unsigned integer data is converted with all fraction bits set to 0. Bitmap data convert to either 0 or 1.

Each fixed-point index is then shifted left by ?GL_INDEX_SHIFT bits, and added to ?GL_INDEX_OFFSET. If ?GL_INDEX_SHIFT is negative, the shift is to the right. In either case, zero bits fill otherwise unspecified bit locations in the result. If ?GL_MAP_STENCIL is true, the index is replaced with the value that it references in lookup table ?GL_PIXEL_MAP_S_TO_S. Whether the lookup replacement of the index is done or not, the integer part of the index is then ANDed with 2 b-1, where b is the number of bits in the stencil buffer. The resulting stencil indices are then written to the stencil buffer such that the nth index is written to location

x n= x r+n% width

y n= y r+|n/width|

where (x r y r) is the current raster position. Only the pixel ownership test, the scissor test, and the stencil writemask affect these write operations.

?GL_DEPTH_COMPONENT: Each pixel is a single-depth component. Floating-point data is converted directly to an internal floating-point format with unspecified precision. Signed integer data is mapped linearly to the internal floating-point format such that the most positive representable integer value maps to 1.0, and the most negative representable value maps to -1.0. Unsigned integer data is mapped similarly: the largest integer value maps to 1.0, and 0 maps to 0.0. The resulting floating-point depth value is then multiplied by ?GL_DEPTH_SCALE and added to ?GL_DEPTH_BIAS. The result is clamped to the range [0 1].

The GL then converts the resulting depth components to fragments by attaching the current raster position color or color index and texture coordinates to each pixel, then assigning x and y window coordinates to the nth fragment such that

x n= x r+n% width

y n= y r+|n/width|

where (x r y r) is the current raster position. These pixel fragments are then treated just like the fragments generated by rasterizing points, lines, or polygons. Texture mapping, fog, and all the fragment operations are applied before the fragments are written to the frame buffer.

?GL_RGBA

?GL_BGRA: Each pixel is a four-component group: For ?GL_RGBA, the red component is first, followed by green, followed by blue, followed by alpha; for ?GL_BGRA the order is blue, green, red and then alpha. Floating-point values are converted directly to an internal floating-point format with unspecified precision. Signed integer values are mapped linearly to the internal floating-point format such that the most positive representable integer value maps to 1.0, and the most negative representable value maps to -1.0. (Note that this mapping does not convert 0 precisely to 0.0.) Unsigned integer data is mapped similarly: The largest integer value maps to 1.0, and 0 maps to 0.0. The resulting floating-point color values are then multiplied by ?GL_c_SCALE and added to ?GL_c_BIAS, where c is RED, GREEN, BLUE, and ALPHA for the respective color components. The results are clamped to the range [0 1].

If ?GL_MAP_COLOR is true, each color component is scaled by the size of lookup table ?GL_PIXEL_MAP_c_TO_c, then replaced by the value that it references in that table. c is R, G, B, or A respectively.

The GL then converts the resulting RGBA colors to fragments by attaching the current raster position z coordinate and texture coordinates to each pixel, then assigning x and y window coordinates to the nth fragment such that

x n= x r+n% width

y n= y r+|n/width|

where (x r y r) is the current raster position. These pixel fragments are then treated just like the fragments generated by rasterizing points, lines, or polygons. Texture mapping, fog, and all the fragment operations are applied before the fragments are written to the frame buffer.

?GL_RED: Each pixel is a single red component. This component is converted to the internal floating-point format in the same way the red component of an RGBA pixel is. It is then converted to an RGBA pixel with green and blue set to 0, and alpha set to 1. After this conversion, the pixel is treated as if it had been read as an RGBA pixel.

?GL_GREEN: Each pixel is a single green component. This component is converted to the internal floating-point format in the same way the green component of an RGBA pixel is. It is then converted to an RGBA pixel with red and blue set to 0, and alpha set to 1. After this conversion, the pixel is treated as if it had been read as an RGBA pixel.

?GL_BLUE: Each pixel is a single blue component. This component is converted to the internal floating-point format in the same way the blue component of an RGBA pixel is. It is then converted to an RGBA pixel with red and green set to 0, and alpha set to 1. After this conversion, the pixel is treated as if it had been read as an RGBA pixel.

?GL_ALPHA: Each pixel is a single alpha component. This component is converted to the internal floating-point format in the same way the alpha component of an RGBA pixel is. It is then converted to an RGBA pixel with red, green, and blue set to 0. After this conversion, the pixel is treated as if it had been read as an RGBA pixel.

?GL_RGB

?GL_BGR: Each pixel is a three-component group: red first, followed by green, followed by blue; for ?GL_BGR, the first component is blue, followed by green and then red. Each component is converted to the internal floating-point format in the same way the red, green, and blue components of an RGBA pixel are. The color triple is converted to an RGBA pixel with alpha set to 1. After this conversion, the pixel is treated as if it had been read as an RGBA pixel.

?GL_LUMINANCE: Each pixel is a single luminance component. This component is converted to the internal floating-point format in the same way the red component of an RGBA pixel is. It is then converted to an RGBA pixel with red, green, and blue set to the converted luminance value, and alpha set to 1. After this conversion, the pixel is treated as if it had been read as an RGBA pixel.

?GL_LUMINANCE_ALPHA: Each pixel is a two-component group: luminance first, followed by alpha. The two components are converted to the internal floating-point format in the same way the red component of an RGBA pixel is. They are then converted to an RGBA pixel with red, green, and blue set to the converted luminance value, and alpha set to the converted alpha value. After this conversion, the pixel is treated as if it had been read as an RGBA pixel.

The following table summarizes the meaning of the valid constants for the type parameter:

TypeCorresponding Type
?GL_UNSIGNED_BYTE unsigned 8-bit integer
?GL_BYTE signed 8-bit integer
?GL_BITMAP single bits in unsigned 8-bit integers
?GL_UNSIGNED_SHORT unsigned 16-bit integer
?GL_SHORT signed 16-bit integer
?GL_UNSIGNED_INT unsigned 32-bit integer
?GL_INT 32-bit integer
?GL_FLOAT single-precision floating-point
?GL_UNSIGNED_BYTE_3_3_2 unsigned 8-bit integer
?GL_UNSIGNED_BYTE_2_3_3_REV unsigned 8-bit integer with reversed component ordering
?GL_UNSIGNED_SHORT_5_6_5 unsigned 16-bit integer
?GL_UNSIGNED_SHORT_5_6_5_REV unsigned 16-bit integer with reversed component ordering
?GL_UNSIGNED_SHORT_4_4_4_4 unsigned 16-bit integer
?GL_UNSIGNED_SHORT_4_4_4_4_REV unsigned 16-bit integer with reversed component ordering
?GL_UNSIGNED_SHORT_5_5_5_1 unsigned 16-bit integer
?GL_UNSIGNED_SHORT_1_5_5_5_REV unsigned 16-bit integer with reversed component ordering
?GL_UNSIGNED_INT_8_8_8_8 unsigned 32-bit integer
?GL_UNSIGNED_INT_8_8_8_8_REV unsigned 32-bit integer with reversed component ordering
?GL_UNSIGNED_INT_10_10_10_2 unsigned 32-bit integer
?GL_UNSIGNED_INT_2_10_10_10_REV unsigned 32-bit integer with reversed component ordering

The rasterization described so far assumes pixel zoom factors of 1. If *gl:pixelZoom/2* is used to change the x and y pixel zoom factors, pixels are converted to fragments as follows. If (x r y r) is the current raster position, and a given pixel is in the nth column and mth row of the pixel rectangle, then fragments are generated for pixels whose centers are in the rectangle with corners at

(x r+(zoom x) n y r+(zoom y) m)

(x r+(zoom x)(n+1) y r+(zoom y)(m+1))

where zoom x is the value of ?GL_ZOOM_X and zoom y is the value of ?GL_ZOOM_Y .

See **external** documentation.

copyPixels(X, Y, Width, Height, Type) -> ok

Types:

X = integer()
Y = integer()
Width = integer()
Height = integer()
Type = enum()

Copy pixels in the frame buffer

gl:copyPixels copies a screen-aligned rectangle of pixels from the specified frame buffer location to a region relative to the current raster position. Its operation is well defined only if the entire pixel source region is within the exposed portion of the window. Results of copies from outside the window, or from regions of the window that are not exposed, are hardware dependent and undefined.

X and Y specify the window coordinates of the lower left corner of the rectangular region to be copied. Width and Height specify the dimensions of the rectangular region to be copied. Both Width and Height must not be negative.

Several parameters control the processing of the pixel data while it is being copied. These parameters are set with three commands: *gl:pixelTransferf/2* , *gl:pixelMapfv/3* , and *gl:pixelZoom/2* . This reference page describes the effects on gl:copyPixels of most, but not all, of the parameters specified by these three commands.

gl:copyPixels copies values from each pixel with the lower left-hand corner at (x+i y+j) for 0<= i< width and 0<= j< height. This pixel is said to be the ith pixel in the jth row. Pixels are copied in row order from the lowest to the highest row, left to right in each row.

Type specifies whether color, depth, or stencil data is to be copied. The details of the transfer for each data type are as follows:

?GL_COLOR: Indices or RGBA colors are read from the buffer currently specified as the read source buffer (see *gl:readBuffer/1*). If the GL is in color index mode, each index that is read from this buffer is converted to a fixed-point format with an unspecified number of bits to the right of the binary point. Each index is then shifted left by ?GL_INDEX_SHIFT bits, and added to ?GL_INDEX_OFFSET. If ?GL_INDEX_SHIFT is negative, the shift is to the right. In either case, zero bits fill otherwise unspecified bit locations in the result. If ?GL_MAP_COLOR is true, the index is replaced with the value that it references in lookup table ?GL_PIXEL_MAP_I_TO_I. Whether the lookup replacement of the index is done or not, the integer part of the index is then ANDed with 2 b-1, where b is the number of bits in a color index buffer.

If the GL is in RGBA mode, the red, green, blue, and alpha components of each pixel that is read are converted to an internal floating-point format with unspecified precision. The conversion maps the largest representable component value to 1.0, and component value 0 to 0.0. The resulting floating-point color values are then multiplied by ?GL_c_SCALE and added to ?GL_c_BIAS, where c is RED, GREEN, BLUE, and ALPHA for the respective color components. The results are clamped to the range [0,1]. If ?GL_MAP_COLOR is true, each color component is scaled by the size of lookup table ?GL_PIXEL_MAP_c_TO_c , then replaced by the value that it references in that table. c is R, G, B, or A.

If the ARB_imaging extension is supported, the color values may be additionally processed by color-table lookups, color-matrix transformations, and convolution filters.

The GL then converts the resulting indices or RGBA colors to fragments by attaching the current raster position z coordinate and texture coordinates to each pixel, then assigning window coordinates (x r+i y r+j), where (x r y r) is the current raster position, and the pixel was the ith pixel in the jth row. These pixel fragments are then treated just like the fragments generated by rasterizing points, lines, or polygons. Texture mapping, fog, and all the fragment operations are applied before the fragments are written to the frame buffer.

?GL_DEPTH: Depth values are read from the depth buffer and converted directly to an internal floating-point format with unspecified precision. The resulting floating-point depth value is then multiplied by ?GL_DEPTH_SCALE and added to ?GL_DEPTH_BIAS . The result is clamped to the range [0,1].

The GL then converts the resulting depth components to fragments by attaching the current raster position color or color index and texture coordinates to each pixel, then assigning window coordinates (x r+i y r+j), where (x r y r) is the current raster position, and the pixel was the ith pixel in the jth row. These pixel fragments are then treated just like the fragments generated by rasterizing points, lines, or polygons. Texture mapping, fog, and all the fragment operations are applied before the fragments are written to the frame buffer.

?GL_STENCIL: Stencil indices are read from the stencil buffer and converted to an internal fixed-point format with an unspecified number of bits to the right of the binary point. Each fixed-point index is then shifted left by ?GL_INDEX_SHIFT bits, and added to ?GL_INDEX_OFFSET. If ?GL_INDEX_SHIFT is negative, the shift is to the right. In either case, zero bits fill otherwise unspecified bit locations in the result. If ?GL_MAP_STENCIL is true, the index is replaced with the value that it references in lookup table ?GL_PIXEL_MAP_S_TO_S. Whether the lookup replacement of the index is done or not, the integer part of the index is then ANDed with 2 b-1, where b is the number of bits in the stencil buffer. The resulting stencil indices are then written to the stencil buffer such that the index read from the ith location of the jth row is written to location (x r+i y r+j), where (x r y r) is the current raster position. Only the pixel ownership test, the scissor test, and the stencil writemask affect these write operations.

The rasterization described thus far assumes pixel zoom factors of 1.0. If *gl:pixelZoom/2* is used to change the x and y pixel zoom factors, pixels are converted to fragments as follows. If (x r y r) is the current raster position, and a given pixel is in the ith location in the jth row of the source pixel rectangle, then fragments are generated for pixels whose centers are in the rectangle with corners at

(x r+(zoom x) i y r+(zoom y) j)

and

(x r+(zoom x)(i+1) y r+(zoom y)(j+1))

where zoom x is the value of ?GL_ZOOM_X and zoom y is the value of ?GL_ZOOM_Y .

See **external** documentation.

stencilFunc(Func, Ref, Mask) -> ok

Types:

```
Func = enum()
Ref = integer()
Mask = integer()
```

Set front and back function and reference value for stencil testing

Stenciling, like depth-buffering, enables and disables drawing on a per-pixel basis. Stencil planes are first drawn into using GL drawing primitives, then geometry and images are rendered using the stencil planes to mask out portions of the screen. Stenciling is typically used in multipass rendering algorithms to achieve special effects, such as decals, outlining, and constructive solid geometry rendering.

The stencil test conditionally eliminates a pixel based on the outcome of a comparison between the reference value and the value in the stencil buffer. To enable and disable the test, call *gl:enable/1* and *gl:enable/1* with argument ?GL_STENCIL_TEST . To specify actions based on the outcome of the stencil test, call *gl:stencilOp/3* or *gl:stencilOpSeparate/4* .

There can be two separate sets of Func , Ref , and Mask parameters; one affects back-facing polygons, and the other affects front-facing polygons as well as other non-polygon primitives. *gl:stencilFunc/3* sets both front and back stencil state to the same values. Use *gl:stencilFuncSeparate/4* to set front and back stencil state to different values.

Func is a symbolic constant that determines the stencil comparison function. It accepts one of eight values, shown in the following list. Ref is an integer reference value that is used in the stencil comparison. It is clamped to the range [0 2 n-1], where n is the number of bitplanes in the stencil buffer. Mask is bitwise ANDed with both the reference value and the stored stencil value, with the ANDed values participating in the comparison.

If stencil represents the value stored in the corresponding stencil buffer location, the following list shows the effect of each comparison function that can be specified by Func . Only if the comparison succeeds is the pixel passed through to the next stage in the rasterization process (see *gl:stencilOp/3*). All tests treat stencil values as unsigned integers in the range [0 2 n-1], where n is the number of bitplanes in the stencil buffer.

The following values are accepted by Func :

?GL_NEVER: Always fails.

?GL_LESS: Passes if (Ref & Mask) < (stencil & Mask).

?GL_LEQUAL: Passes if (Ref & Mask) <= (stencil & Mask).

?GL_GREATER: Passes if (Ref & Mask) > (stencil & Mask).

?GL_GEQUAL: Passes if (Ref & Mask) >= (stencil & Mask).

?GL_EQUAL: Passes if (Ref & Mask) = (stencil & Mask).

?GL_NOTEQUAL: Passes if (Ref & Mask) != (stencil & Mask).

?GL_ALWAYS: Always passes.

See **external** documentation.

stencilMask(Mask) -> ok

Types:

Mask = integer()

Control the front and back writing of individual bits in the stencil planes

gl:stencilMask controls the writing of individual bits in the stencil planes. The least significant n bits of Mask , where n is the number of bits in the stencil buffer, specify a mask. Where a 1 appears in the mask, it's possible to write to the corresponding bit in the stencil buffer. Where a 0 appears, the corresponding bit is write-protected. Initially, all bits are enabled for writing.

There can be two separate Mask writemasks; one affects back-facing polygons, and the other affects front-facing polygons as well as other non-polygon primitives. *gl:stencilMask/1* sets both front and back stencil writemasks to the same values. Use *gl:stencilMaskSeparate/2* to set front and back stencil writemasks to different values.

See **external** documentation.

stencilOp(Fail, Zfail, Zpass) -> ok

Types:

Fail = enum()

Zfail = enum()

Zpass = enum()

Set front and back stencil test actions

Stenciling, like depth-buffering, enables and disables drawing on a per-pixel basis. You draw into the stencil planes using GL drawing primitives, then render geometry and images, using the stencil planes to mask out portions of

the screen. Stenciling is typically used in multipass rendering algorithms to achieve special effects, such as decals, outlining, and constructive solid geometry rendering.

The stencil test conditionally eliminates a pixel based on the outcome of a comparison between the value in the stencil buffer and a reference value. To enable and disable the test, call *gl:enable/1* and *gl:enable/1* with argument `?GL_STENCIL_TEST` ; to control it, call *gl:stencilFunc/3* or *gl:stencilFuncSeparate/4* .

There can be two separate sets of `Sfail` , `Dpfail` , and `Dppass` parameters; one affects back-facing polygons, and the other affects front-facing polygons as well as other non-polygon primitives. *gl:stencilOp/3* sets both front and back stencil state to the same values. Use *gl:stencilOpSeparate/4* to set front and back stencil state to different values.

`gl:stencilOp` takes three arguments that indicate what happens to the stored stencil value while stenciling is enabled. If the stencil test fails, no change is made to the pixel's color or depth buffers, and `Sfail` specifies what happens to the stencil buffer contents. The following eight actions are possible.

`?GL_KEEP`: Keeps the current value.

`?GL_ZERO`: Sets the stencil buffer value to 0.

`?GL_REPLACE`: Sets the stencil buffer value to `ref`, as specified by *gl:stencilFunc/3* .

`?GL_INCR`: Increments the current stencil buffer value. Clamps to the maximum representable unsigned value.

`?GL_INCR_WRAP`: Increments the current stencil buffer value. Wraps stencil buffer value to zero when incrementing the maximum representable unsigned value.

`?GL_DECR`: Decrements the current stencil buffer value. Clamps to 0.

`?GL_DECR_WRAP`: Decrements the current stencil buffer value. Wraps stencil buffer value to the maximum representable unsigned value when decrementing a stencil buffer value of zero.

`?GL_INVERT`: Bitwise inverts the current stencil buffer value.

Stencil buffer values are treated as unsigned integers. When incremented and decremented, values are clamped to 0 and 2 n-1, where n is the value returned by querying `?GL_STENCIL_BITS` .

The other two arguments to `gl:stencilOp` specify stencil buffer actions that depend on whether subsequent depth buffer tests succeed (`Dppass`) or fail (`Dpfail`) (see *gl:depthFunc/1*). The actions are specified using the same eight symbolic constants as `Sfail` . Note that `Dpfail` is ignored when there is no depth buffer, or when the depth buffer is not enabled. In these cases, `Sfail` and `Dppass` specify stencil action when the stencil test fails and passes, respectively.

See **external** documentation.

clearStencil(S) -> ok

Types:

S = integer()

Specify the clear value for the stencil buffer

`gl:clearStencil` specifies the index used by *gl:clear/1* to clear the stencil buffer. `S` is masked with 2 m-1, where m is the number of bits in the stencil buffer.

See **external** documentation.

texGend(Coord, Pname, Param) -> ok

Types:

Coord = enum()
Pname = enum()
Param = float()

Control the generation of texture coordinates

gl:texGen selects a texture-coordinate generation function or supplies coefficients for one of the functions. Coord names one of the (s, t, r, q) texture coordinates; it must be one of the symbols ?GL_S, ?GL_T, ?GL_R , or ?GL_Q. Pname must be one of three symbolic constants: ?GL_TEXTURE_GEN_MODE , ?GL_OBJECT_PLANE, or ?GL_EYE_PLANE. If Pname is ?GL_TEXTURE_GEN_MODE , then Params chooses a mode, one of ?GL_OBJECT_LINEAR, ?GL_EYE_LINEAR , ?GL_SPHERE_MAP, ?GL_NORMAL_MAP, or ?GL_REFLECTION_MAP. If Pname is either ?GL_OBJECT_PLANE or ?GL_EYE_PLANE, Params contains coefficients for the corresponding texture generation function.

If the texture generation function is ?GL_OBJECT_LINEAR, the function

g= p 1*x o+p 2*y o+p 3*z o+p 4*w o

is used, where g is the value computed for the coordinate named in Coord , p 1, p 2, p 3, and p 4 are the four values supplied in Params , and x o, y o, z o, and w o are the object coordinates of the vertex. This function can be used, for example, to texture-map terrain using sea level as a reference plane (defined by p 1, p 2, p 3, and p 4). The altitude of a terrain vertex is computed by the ?GL_OBJECT_LINEAR coordinate generation function as its distance from sea level; that altitude can then be used to index the texture image to map white snow onto peaks and green grass onto foothills.

If the texture generation function is ?GL_EYE_LINEAR, the function

g=(p 1)"*x e+(p 2)"*y e+(p 3)"*z e+(p 4)"*w e

is used, where

((p 1)" (p 2)" (p 3)" (p 4)")=(p 1 p 2 p 3 p 4) M -1

and x e, y e, z e, and w e are the eye coordinates of the vertex, p 1, p 2, p 3, and p 4 are the values supplied in Params , and M is the modelview matrix when gl:texGen is invoked. If M is poorly conditioned or singular, texture coordinates generated by the resulting function may be inaccurate or undefined.

Note that the values in Params define a reference plane in eye coordinates. The modelview matrix that is applied to them may not be the same one in effect when the polygon vertices are transformed. This function establishes a field of texture coordinates that can produce dynamic contour lines on moving objects.

If the texture generation function is ?GL_SPHERE_MAP and Coord is either ?GL_S or ?GL_T, s and t texture coordinates are generated as follows. Let u be the unit vector pointing from the origin to the polygon vertex (in eye coordinates). Let n sup prime be the current normal, after transformation to eye coordinates. Let

f=(f x f y f z) T be the reflection vector such that

f= u-2 n" (n") T u

Finally, let m= 2 ((f x) 2+(f y) 2+(f z+1) 2). Then the values assigned to the s and t texture coordinates are

s= f x/m+1/2

t= f y/m+1/2

To enable or disable a texture-coordinate generation function, call *gl:enable/1* or *gl:enable/1* with one of the symbolic texture-coordinate names (?GL_TEXTURE_GEN_S , ?GL_TEXTURE_GEN_T, ?GL_TEXTURE_GEN_R, or ?GL_TEXTURE_GEN_Q) as the argument. When enabled, the specified texture coordinate is computed according to the generating function associated with that coordinate. When disabled, subsequent vertices take the specified texture coordinate from the current set of texture coordinates. Initially, all texture generation functions are set to ?GL_EYE_LINEAR and are disabled. Both s plane equations are (1, 0, 0, 0), both t plane equations are (0, 1, 0, 0), and all r and q plane equations are (0, 0, 0, 0).

When the ARB_multitexture extension is supported, gl:texGen sets the texture generation parameters for the currently active texture unit, selected with *gl:activeTexture/1* .

See **external** documentation.

texGenf(Coord, Pname, Param) -> ok

Types:

Coord = enum()
Pname = enum()
Param = float()

See *texGend/3*

texGeni(Coord, Pname, Param) -> ok

Types:

Coord = enum()
Pname = enum()
Param = integer()

See *texGend/3*

texGendv(Coord, Pname, Params) -> ok

Types:

Coord = enum()
Pname = enum()
Params = {float()}

See *texGend/3*

texGenfv(Coord, Pname, Params) -> ok

Types:

Coord = enum()
Pname = enum()
Params = {float()}

See *texGend/3*

texGeniv(Coord, Pname, Params) -> ok

Types:

Coord = enum()
Pname = enum()
Params = {integer()}

See *texGend/3*

getTexGendv(Coord, Pname) -> {float(), float(), float(), float()}

Types:

Coord = enum()
Pname = enum()

Return texture coordinate generation parameters

`gl:getTexGen` returns in `Params` selected parameters of a texture coordinate generation function that was specified using *gl:texGend/3* . `Coord` names one of the (`s`, `t`, `r`, `q`) texture coordinates, using the symbolic constant `?GL_S`, `?GL_T`, `?GL_R`, or `?GL_Q`.

Pname specifies one of three symbolic names:

?GL_TEXTURE_GEN_MODE: Params returns the single-valued texture generation function, a symbolic constant. The initial value is ?GL_EYE_LINEAR.

?GL_OBJECT_PLANE: Params returns the four plane equation coefficients that specify object linear-coordinate generation. Integer values, when requested, are mapped directly from the internal floating-point representation.

?GL_EYE_PLANE: Params returns the four plane equation coefficients that specify eye linear-coordinate generation. Integer values, when requested, are mapped directly from the internal floating-point representation. The returned values are those maintained in eye coordinates. They are not equal to the values specified using *gl:texGend/3* , unless the modelview matrix was identity when *gl:texGend/3* was called.

See **external** documentation.

getTexGenfv(Coord, Pname) -> {float(), float(), float(), float()}

Types:

Coord = enum()
Pname = enum()

See *getTexGendv/2*

getTexGeniv(Coord, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Coord = enum()
Pname = enum()

See *getTexGendv/2*

texEnvf(Target, Pname, Param) -> ok

Types:

Target = enum()
Pname = enum()
Param = float()

glTexEnvf

See **external** documentation.

texEnvi(Target, Pname, Param) -> ok

Types:

Target = enum()
Pname = enum()
Param = integer()

glTexEnvi

See **external** documentation.

texEnvfv(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()

Params = {float()}

Set texture environment parameters

A texture environment specifies how texture values are interpreted when a fragment is textured. When Target is ?GL_TEXTURE_FILTER_CONTROL, Pname must be ?GL_TEXTURE_LOD_BIAS . When Target is ?GL_TEXTURE_ENV, Pname can be ?GL_TEXTURE_ENV_MODE , ?GL_TEXTURE_ENV_COLOR, ?GL_COMBINE_RGB, ?GL_COMBINE_ALPHA, ?GL_RGB_SCALE , ?GL_ALPHA_SCALE, ?GL_SRC0_RGB, ?GL_SRC1_RGB, ?GL_SRC2_RGB, ?GL_SRC0_ALPHA , ?GL_SRC1_ALPHA, or ?GL_SRC2_ALPHA.

If Pname is ?GL_TEXTURE_ENV_MODE, then Params is (or points to) the symbolic name of a texture function. Six texture functions may be specified: ?GL_ADD , ?GL_MODULATE, ?GL_DECAL, ?GL_BLEND, ?GL_REPLACE, or ?GL_COMBINE .

The following table shows the correspondence of filtered texture values R t, G t, B t, A t, L t, I t to texture source components. C s and A s are used by the texture functions described below.

Texture Base Internal Format C s A s
?GL_ALPHA (0, 0, 0) A t
?GL_LUMINANCE (L t, L t, L t) 1
?GL_LUMINANCE_ALPHA (L t, L t, L t) A t
?GL_INTENSITY (I t, I t, I t) I t
?GL_RGB (R t, G t, B t) 1
?GL_RGBA (R t, G t, B t) A t

A texture function acts on the fragment to be textured using the texture image value that applies to the fragment (see *gl:texParameterf/3*) and produces an RGBA color for that fragment. The following table shows how the RGBA color is produced for each of the first five texture functions that can be chosen. C is a triple of color values (RGB) and A is the associated alpha value. RGBA values extracted from a texture image are in the range [0,1]. The subscript p refers to the color computed from the previous texture stage (or the incoming fragment if processing texture stage 0), the subscript s to the texture source color, the subscript c to the texture environment color, and the subscript v indicates a value produced by the texture function.

Texture Base Internal Format ?Value?GL_REPLACE Function ?GL_MODULATE Function ?GL_DECAL Function ?GL_BLEND Function ?GL_ADD Function
?GL_ALPHA C v= C p C p undefined C p C p
A v= A s A p A s A v= A p A s A p A s
?GL_LUMINANCE C v= C s C p C s undefined C p (1-C s)+C c C s C p+C s
(or 1) A v= A p A p A p A p
?GL_LUMINANCE_ALPHA C v= C s C p C s undefined C p (1-C s)+C c C s C p+C s
(or 2) A v= A s A p A s A p A s A p A s
?GL_INTENSITY C v= C s C p C s undefined C p (1-C s)+C c C s C p+C s
A v= A s A p A s A p (1-A s)+A c A s A p+A s
?GL_RGB C v= C s C p C s C s C p (1-C s)+C c C s C p+C s
(or 3) A v= A p A p A p A p A p
?GL_RGBA C v= C s C p C s C p (1-A s)+C s A s C p (1-C s)+C c C s C p+C s
(or 4) A v= A s A p A s A p A p A s A p A s

If Pname is ?GL_TEXTURE_ENV_MODE, and Params is ?GL_COMBINE, the form of the texture function depends on the values of ?GL_COMBINE_RGB and ?GL_COMBINE_ALPHA .

The following describes how the texture sources, as specified by ?GL_SRC0_RGB, ?GL_SRC1_RGB , ?GL_SRC2_RGB, ?GL_SRC0_ALPHA, ?GL_SRC1_ALPHA, and ?GL_SRC2_ALPHA , are combined to produce a final texture color. In the following tables, ?GL_SRC0_c is represented by Arg0, ?GL_SRC1_c is represented by Arg1, and ?GL_SRC2_c is represented by Arg2.

?GL_COMBINE_RGB accepts any of ?GL_REPLACE, ?GL_MODULATE, ?GL_ADD , ?GL_ADD_SIGNED, ?GL_INTERPOLATE, ?GL_SUBTRACT, ?GL_DOT3_RGB, or ?GL_DOT3_RGBA.

?GL_COMBINE_RGBTexture Function
?GL_REPLACE Arg0
?GL_MODULATE Arg0*Arg1
?GL_ADD Arg0+Arg1
?GL_ADD_SIGNED Arg0+Arg1-0.5
?GL_INTERPOLATE Arg0*Arg2+Arg1*(1- Arg2)
?GL_SUBTRACT Arg0-Arg1
?GL_DOT3_RGB or ?GL_DOT3_RGBA 4*((((Arg0 r)-0.5)*((Arg1 r)-0.5))+(((Arg0 g)-0.5)*((Arg1 g)-0.5))+(((Arg0 b)-0.5)*((Arg1 b)-0.5)))

The scalar results for ?GL_DOT3_RGB and ?GL_DOT3_RGBA are placed into each of the 3 (RGB) or 4 (RGBA) components on output.

Likewise, ?GL_COMBINE_ALPHA accepts any of ?GL_REPLACE, ?GL_MODULATE, ?GL_ADD, ?GL_ADD_SIGNED, ?GL_INTERPOLATE, or ?GL_SUBTRACT. The following table describes how alpha values are combined:

?GL_COMBINE_ALPHATexture Function
?GL_REPLACE Arg0
?GL_MODULATE Arg0*Arg1
?GL_ADD Arg0+Arg1
?GL_ADD_SIGNED Arg0+Arg1-0.5
?GL_INTERPOLATE Arg0*Arg2+Arg1*(1- Arg2)
?GL_SUBTRACT Arg0-Arg1

In the following tables, the value C s represents the color sampled from the currently bound texture, C c represents the constant texture-environment color, C f represents the primary color of the incoming fragment, and C p represents the color computed from the previous texture stage or C f if processing texture stage 0. Likewise, A s, A c, A f, and A p represent the respective alpha values.

The following table describes the values assigned to Arg0, Arg1, and Arg2 based upon the RGB sources and operands:

?GL_SRCn_RGB?GL_OPERANDn_RGBArgument Value
?GL_TEXTURE?GL_SRC_COLOR(C s)
?GL_ONE_MINUS_SRC_COLOR 1-(C s)
?GL_SRC_ALPHA(A s)
?GL_ONE_MINUS_SRC_ALPHA 1-(A s)
?GL_TEXTUREn?GL_SRC_COLOR(C s)
?GL_ONE_MINUS_SRC_COLOR 1-(C s)
?GL_SRC_ALPHA (A s)
?GL_ONE_MINUS_SRC_ALPHA 1-(A s)
?GL_CONSTANT?GL_SRC_COLOR(C c)
?GL_ONE_MINUS_SRC_COLOR 1-(C c)
?GL_SRC_ALPHA(A c)
?GL_ONE_MINUS_SRC_ALPHA 1-(A c)
?GL_PRIMARY_COLOR ?GL_SRC_COLOR(C f)
?GL_ONE_MINUS_SRC_COLOR 1-(C f)
?GL_SRC_ALPHA(A f)
?GL_ONE_MINUS_SRC_ALPHA 1-(A f)
?GL_PREVIOUS?GL_SRC_COLOR (C p)
?GL_ONE_MINUS_SRC_COLOR 1-(C p)
?GL_SRC_ALPHA(A p)
?GL_ONE_MINUS_SRC_ALPHA 1-(A p)

For ?GL_TEXTUREn sources, C s and A s represent the color and alpha, respectively, produced from texture stage n.

The follow table describes the values assigned to Arg0, Arg1, and Arg2 based upon the alpha sources and operands:

?GL_SRCn_ALPHA?GL_OPERANDn_ALPHAArgument Value
?GL_TEXTURE?GL_SRC_ALPHA(A s)
?GL_ONE_MINUS_SRC_ALPHA 1-(A s)
?GL_TEXTUREn ?GL_SRC_ALPHA(A s)
?GL_ONE_MINUS_SRC_ALPHA 1-(A s)
?GL_CONSTANT?GL_SRC_ALPHA(A c)
?GL_ONE_MINUS_SRC_ALPHA 1-(A c)
?GL_PRIMARY_COLOR ?GL_SRC_ALPHA(A f)
?GL_ONE_MINUS_SRC_ALPHA 1-(A f)
?GL_PREVIOUS?GL_SRC_ALPHA(A p)
?GL_ONE_MINUS_SRC_ALPHA 1-(A p)

The RGB and alpha results of the texture function are multipled by the values of ?GL_RGB_SCALE and ?GL_ALPHA_SCALE, respectively, and clamped to the range [0 1].

If Pname is ?GL_TEXTURE_ENV_COLOR, Params is a pointer to an array that holds an RGBA color consisting of four values. Integer color components are interpreted linearly such that the most positive integer maps to 1.0, and the most negative integer maps to -1.0. The values are clamped to the range [0,1] when they are specified. C c takes these four values.

If Pname is ?GL_TEXTURE_LOD_BIAS, the value specified is added to the texture level-of-detail parameter, that selects which mipmap, or mipmaps depending upon the selected ?GL_TEXTURE_MIN_FILTER, will be sampled.

?GL_TEXTURE_ENV_MODE defaults to ?GL_MODULATE and ?GL_TEXTURE_ENV_COLOR defaults to (0, 0, 0, 0).

If Target is ?GL_POINT_SPRITE and Pname is ?GL_COORD_REPLACE, the boolean value specified is used to either enable or disable point sprite texture coordinate replacement. The default value is ?GL_FALSE.

See **external** documentation.

texEnviv(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {integer()}

See *texEnvfv/3*

getTexEnvfv(Target, Pname) -> {float(), float(), float(), float()}

Types:

Target = enum()
Pname = enum()

Return texture environment parameters

gl:getTexEnv returns in Params selected values of a texture environment that was specified with *gl:texEnvfv/3* . Target specifies a texture environment.

When Target is ?GL_TEXTURE_FILTER_CONTROL, Pname must be ?GL_TEXTURE_LOD_BIAS . When Target is ?GL_POINT_SPRITE, Pname must be ?GL_COORD_REPLACE . When Target is ?GL_TEXTURE_ENV, Pname can be ?GL_TEXTURE_ENV_MODE , ?GL_TEXTURE_ENV_COLOR, ?GL_COMBINE_RGB, ?GL_COMBINE_ALPHA, ?GL_RGB_SCALE , ?GL_ALPHA_SCALE, ?GL_SRC0_RGB, ?GL_SRC1_RGB, ?GL_SRC2_RGB, ?GL_SRC0_ALPHA, ?GL_SRC1_ALPHA, or ?GL_SRC2_ALPHA.

Pname names a specific texture environment parameter, as follows:

?GL_TEXTURE_ENV_MODE: Params returns the single-valued texture environment mode, a symbolic constant. The initial value is ?GL_MODULATE.

?GL_TEXTURE_ENV_COLOR: Params returns four integer or floating-point values that are the texture environment color. Integer values, when requested, are linearly mapped from the internal floating-point representation such that 1.0 maps to the most positive representable integer, and -1.0 maps to the most negative representable integer. The initial value is (0, 0, 0, 0).

?GL_TEXTURE_LOD_BIAS: Params returns a single floating-point value that is the texture level-of-detail bias. The initial value is 0.

?GL_COMBINE_RGB: Params returns a single symbolic constant value representing the current RGB combine mode. The initial value is ?GL_MODULATE.

?GL_COMBINE_ALPHA: Params returns a single symbolic constant value representing the current alpha combine mode. The initial value is ?GL_MODULATE.

?GL_SRC0_RGB: Params returns a single symbolic constant value representing the texture combiner zero's RGB source. The initial value is ?GL_TEXTURE.

?GL_SRC1_RGB: Params returns a single symbolic constant value representing the texture combiner one's RGB source. The initial value is ?GL_PREVIOUS.

?GL_SRC2_RGB: Params returns a single symbolic constant value representing the texture combiner two's RGB source. The initial value is ?GL_CONSTANT.

?GL_SRC0_ALPHA: Params returns a single symbolic constant value representing the texture combiner zero's alpha source. The initial value is ?GL_TEXTURE.

?GL_SRC1_ALPHA: Params returns a single symbolic constant value representing the texture combiner one's alpha source. The initial value is ?GL_PREVIOUS.

?GL_SRC2_ALPHA: Params returns a single symbolic constant value representing the texture combiner two's alpha source. The initial value is ?GL_CONSTANT.

?GL_OPERAND0_RGB: Params returns a single symbolic constant value representing the texture combiner zero's RGB operand. The initial value is ?GL_SRC_COLOR.

?GL_OPERAND1_RGB: Params returns a single symbolic constant value representing the texture combiner one's RGB operand. The initial value is ?GL_SRC_COLOR.

?GL_OPERAND2_RGB: Params returns a single symbolic constant value representing the texture combiner two's RGB operand. The initial value is ?GL_SRC_ALPHA.

?GL_OPERAND0_ALPHA: Params returns a single symbolic constant value representing the texture combiner zero's alpha operand. The initial value is ?GL_SRC_ALPHA.

?GL_OPERAND1_ALPHA: Params returns a single symbolic constant value representing the texture combiner one's alpha operand. The initial value is ?GL_SRC_ALPHA.

?GL_OPERAND2_ALPHA: Params returns a single symbolic constant value representing the texture combiner two's alpha operand. The initial value is ?GL_SRC_ALPHA.

?GL_RGB_SCALE: Params returns a single floating-point value representing the current RGB texture combiner scaling factor. The initial value is 1.0.

?GL_ALPHA_SCALE: Params returns a single floating-point value representing the current alpha texture combiner scaling factor. The initial value is 1.0.

?GL_COORD_REPLACE: Params returns a single boolean value representing the current point sprite texture coordinate replacement enable state. The initial value is ?GL_FALSE .

See **external** documentation.

getTexEnviv(Target, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Target = enum()

Pname = enum()

See *getTexEnvfv/2*

texParameterf(Target, Pname, Param) -> ok

Types:

Target = enum()

Pname = enum()

Param = float()

Set texture parameters

`gl:texParameter` assigns the value or values in `Params` to the texture parameter specified as `Pname` . `Target` defines the target texture, either `?GL_TEXTURE_1D` , `?GL_TEXTURE_2D`, `?GL_TEXTURE_1D_ARRAY`, `?GL_TEXTURE_2D_ARRAY`, `?GL_TEXTURE_RECTANGLE` , or `?GL_TEXTURE_3D`. The following symbols are accepted in `Pname` :

`?GL_TEXTURE_BASE_LEVEL`: Specifies the index of the lowest defined mipmap level. This is an integer value. The initial value is 0.

`?GL_TEXTURE_BORDER_COLOR`: The data in `Params` specifies four values that define the border values that should be used for border texels. If a texel is sampled from the border of the texture, the values of `?GL_TEXTURE_BORDER_COLOR` are interpreted as an RGBA color to match the texture's internal format and substituted for the non-existent texel data. If the texture contains depth components, the first component of `?GL_TEXTURE_BORDER_COLOR` is interpreted as a depth value. The initial value is (0.0, 0.0, 0.0, 0.0).

If the values for `?GL_TEXTURE_BORDER_COLOR` are specified with `gl:texParameterIiv` or `gl:texParameterIuiv`, the values are stored unmodified with an internal data type of integer. If specified with `gl:texParameteriv`, they are converted to floating point with the following equation: f= 2 c+1 2 b-/1. If specified with `gl:texParameterfv` , they are stored unmodified as floating-point values.

`?GL_TEXTURE_COMPARE_FUNC`: Specifies the comparison operator used when `?GL_TEXTURE_COMPARE_MODE` is set to `?GL_COMPARE_REF_TO_TEXTURE`. Permissible values are:

`Texture Comparison FunctionComputed result`
`?GL_LEQUAL` result={1.0 0.0   r<=(D t) r>(D t))
`?GL_GEQUAL` result={1.0 0.0   r>=(D t) r<(D t))
`?GL_LESS` result={1.0 0.0   r< (D t) r>=(D t))
`?GL_GREATER` result={1.0 0.0   r>(D t) r<=(D t))
`?GL_EQUAL` result={1.0 0.0   r=(D t) r≠(D t))
`?GL_NOTEQUAL` result={1.0 0.0   r≠(D t) r=(D t))
`?GL_ALWAYS` result= 1.0
`?GL_NEVER` result= 0.0

where r is the current interpolated texture coordinate, and D t is the depth texture value sampled from the currently bound depth texture. result is assigned to the the red channel.

`?GL_TEXTURE_COMPARE_MODE`: Specifies the texture comparison mode for currently bound depth textures. That is, a texture whose internal format is `?GL_DEPTH_COMPONENT_*` ; see *gl:texImage2D/9*) Permissible values are:

`?GL_COMPARE_REF_TO_TEXTURE`: Specifies that the interpolated and clamped r texture coordinate should be compared to the value in the currently bound depth texture. See the discussion of `?GL_TEXTURE_COMPARE_FUNC` for details of how the comparison is evaluated. The result of the comparison is assigned to the red channel.

?GL_NONE: Specifies that the red channel should be assigned the appropriate value from the currently bound depth texture.

?GL_TEXTURE_LOD_BIAS: Params specifies a fixed bias value that is to be added to the level-of-detail parameter for the texture before texture sampling. The specified value is added to the shader-supplied bias value (if any) and subsequently clamped into the implementation-defined range [(- bias max)(bias max)], where bias max is the value of the implementation defined constant ?GL_MAX_TEXTURE_LOD_BIAS. The initial value is 0.0.

?GL_TEXTURE_MIN_FILTER: The texture minifying function is used whenever the level-of-detail function used when sampling from the texture determines that the texture should be minified. There are six defined minifying functions. Two of them use either the nearest texture elements or a weighted average of multiple texture elements to compute the texture value. The other four use mipmaps.

A mipmap is an ordered set of arrays representing the same image at progressively lower resolutions. If the texture has dimensions 2 n*2 m, there are max(n m)+1 mipmaps. The first mipmap is the original texture, with dimensions 2 n*2 m. Each subsequent mipmap has dimensions 2(k-1)*2(l-1), where 2 k*2 l are the dimensions of the previous mipmap, until either k= 0 or l= 0. At that point, subsequent mipmaps have dimension 1*2(l-1) or 2(k-1)*1 until the final mipmap, which has dimension 1*1. To define the mipmaps, call *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:copyTexImage1D/7* , or *gl:copyTexImage2D/8* with the level argument indicating the order of the mipmaps. Level 0 is the original texture; level max(n m) is the final 1*1 mipmap.

Params supplies a function for minifying the texture as one of the following:

?GL_NEAREST: Returns the value of the texture element that is nearest (in Manhattan distance) to the specified texture coordinates.

?GL_LINEAR: Returns the weighted average of the four texture elements that are closest to the specified texture coordinates. These can include items wrapped or repeated from other parts of a texture, depending on the values of ?GL_TEXTURE_WRAP_S and ?GL_TEXTURE_WRAP_T , and on the exact mapping.

?GL_NEAREST_MIPMAP_NEAREST: Chooses the mipmap that most closely matches the size of the pixel being textured and uses the ?GL_NEAREST criterion (the texture element closest to the specified texture coordinates) to produce a texture value.

?GL_LINEAR_MIPMAP_NEAREST: Chooses the mipmap that most closely matches the size of the pixel being textured and uses the ?GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the specified texture coordinates) to produce a texture value.

?GL_NEAREST_MIPMAP_LINEAR: Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the ?GL_NEAREST criterion (the texture element closest to the specified texture coordinates) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

?GL_LINEAR_MIPMAP_LINEAR: Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the ?GL_LINEAR criterion (a weighted average of the texture elements that are closest to the specified texture coordinates) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

As more texture elements are sampled in the minification process, fewer aliasing artifacts will be apparent. While the ?GL_NEAREST and ?GL_LINEAR minification functions can be faster than the other four, they sample only one or multiple texture elements to determine the texture value of the pixel being rendered and can produce moire patterns or ragged transitions. The initial value of ?GL_TEXTURE_MIN_FILTER is ?GL_NEAREST_MIPMAP_LINEAR .

?GL_TEXTURE_MAG_FILTER: The texture magnification function is used whenever the level-of-detail function used when sampling from the texture determines that the texture should be magified. It sets the texture magnification function to either ?GL_NEAREST or ?GL_LINEAR (see below). ?GL_NEAREST is generally faster than ?GL_LINEAR , but it can produce textured images with sharper edges because the transition between texture elements is not as smooth. The initial value of ?GL_TEXTURE_MAG_FILTER is ?GL_LINEAR .

?GL_NEAREST: Returns the value of the texture element that is nearest (in Manhattan distance) to the specified texture coordinates.

?GL_LINEAR: Returns the weighted average of the texture elements that are closest to the specified texture coordinates. These can include items wrapped or repeated from other parts of a texture, depending on the values of ?GL_TEXTURE_WRAP_S and ?GL_TEXTURE_WRAP_T , and on the exact mapping.

?GL_TEXTURE_MIN_LOD: Sets the minimum level-of-detail parameter. This floating-point value limits the selection of highest resolution mipmap (lowest mipmap level). The initial value is -1000.

?GL_TEXTURE_MAX_LOD: Sets the maximum level-of-detail parameter. This floating-point value limits the selection of the lowest resolution mipmap (highest mipmap level). The initial value is 1000.

?GL_TEXTURE_MAX_LEVEL: Sets the index of the highest defined mipmap level. This is an integer value. The initial value is 1000.

?GL_TEXTURE_SWIZZLE_R: Sets the swizzle that will be applied to the r component of a texel before it is returned to the shader. Valid values for Param are ?GL_RED , ?GL_GREEN, ?GL_BLUE, ?GL_ALPHA, ?GL_ZERO and ?GL_ONE. If ?GL_TEXTURE_SWIZZLE_R is ?GL_RED, the value for r will be taken from the first channel of the fetched texel. If ?GL_TEXTURE_SWIZZLE_R is ?GL_GREEN , the value for r will be taken from the second channel of the fetched texel. If ?GL_TEXTURE_SWIZZLE_R is ?GL_BLUE, the value for r will be taken from the third channel of the fetched texel. If ?GL_TEXTURE_SWIZZLE_R is ?GL_ALPHA, the value for r will be taken from the fourth channel of the fetched texel. If ?GL_TEXTURE_SWIZZLE_R is ?GL_ZERO , the value for r will be subtituted with 0.0. If ?GL_TEXTURE_SWIZZLE_R is ?GL_ONE , the value for r will be subtituted with 1.0. The initial value is ?GL_RED.

?GL_TEXTURE_SWIZZLE_G: Sets the swizzle that will be applied to the g component of a texel before it is returned to the shader. Valid values for Param and their effects are similar to those of ?GL_TEXTURE_SWIZZLE_R. The initial value is ?GL_GREEN .

?GL_TEXTURE_SWIZZLE_B: Sets the swizzle that will be applied to the b component of a texel before it is returned to the shader. Valid values for Param and their effects are similar to those of ?GL_TEXTURE_SWIZZLE_R. The initial value is ?GL_BLUE .

?GL_TEXTURE_SWIZZLE_A: Sets the swizzle that will be applied to the a component of a texel before it is returned to the shader. Valid values for Param and their effects are similar to those of ?GL_TEXTURE_SWIZZLE_R. The initial value is ?GL_ALPHA .

?GL_TEXTURE_SWIZZLE_RGBA: Sets the swizzles that will be applied to the r, g, b, and a components of a texel before they are returned to the shader. Valid values for Params and their effects are similar to those of ?GL_TEXTURE_SWIZZLE_R, except that all channels are specified simultaneously. Setting the value of ?GL_TEXTURE_SWIZZLE_RGBA is equivalent (assuming no errors are generated) to setting the parameters of each of ?GL_TEXTURE_SWIZZLE_R , ?GL_TEXTURE_SWIZZLE_G, ?GL_TEXTURE_SWIZZLE_B, and ?GL_TEXTURE_SWIZZLE_A successively.

?GL_TEXTURE_WRAP_S: Sets the wrap parameter for texture coordinate s to either ?GL_CLAMP_TO_EDGE , ?GL_CLAMP_TO_BORDER, ?GL_MIRRORED_REPEAT, or ?GL_REPEAT. ?GL_CLAMP_TO_EDGE causes s coordinates to be clamped to the range [(1 2/N) 1-(1 2/N)], where N is the size of the texture in the direction of clamping. ?GL_CLAMP_TO_BORDER evaluates s coordinates in a similar manner to ?GL_CLAMP_TO_EDGE. However, in cases where clamping would have occurred in ?GL_CLAMP_TO_EDGE mode, the fetched texel data is substituted with the values specified by ?GL_TEXTURE_BORDER_COLOR. ?GL_REPEAT causes the integer part of the s coordinate to be ignored; the GL uses only the fractional part, thereby creating a repeating pattern. ?GL_MIRRORED_REPEAT causes the s coordinate to be set to the fractional part of the texture coordinate if the integer part of s is even; if the integer part of s is odd, then the s texture coordinate is set to 1- frac(s), where frac(s) represents the fractional part of s. Initially, ?GL_TEXTURE_WRAP_S is set to ?GL_REPEAT.

?GL_TEXTURE_WRAP_T: Sets the wrap parameter for texture coordinate t to either ?GL_CLAMP_TO_EDGE , ?GL_CLAMP_TO_BORDER, ?GL_MIRRORED_REPEAT, or ?GL_REPEAT. See the discussion under ?GL_TEXTURE_WRAP_S. Initially, ?GL_TEXTURE_WRAP_T is set to ?GL_REPEAT.

?GL_TEXTURE_WRAP_R: Sets the wrap parameter for texture coordinate r to either ?GL_CLAMP_TO_EDGE , ?GL_CLAMP_TO_BORDER, ?GL_MIRRORED_REPEAT, or ?GL_REPEAT. See the discussion under ? GL_TEXTURE_WRAP_S. Initially, ?GL_TEXTURE_WRAP_R is set to ?GL_REPEAT.

See **external** documentation.

texParameteri(Target, Pname, Param) -> ok

Types:

Target = enum()
Pname = enum()
Param = integer()

See *texParameterf/3*

texParameterfv(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {float()}

See *texParameterf/3*

texParameteriv(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {integer()}

See *texParameterf/3*

getTexParameterfv(Target, Pname) -> {float(), float(), float(), float()}

Types:

Target = enum()
Pname = enum()

Return texture parameter values

gl:getTexParameter returns in Params the value or values of the texture parameter specified as Pname . Target defines the target texture. ?GL_TEXTURE_1D, ?GL_TEXTURE_2D, ? GL_TEXTURE_3D, ?GL_TEXTURE_1D_ARRAY, ?GL_TEXTURE_2D_ARRAY , ?GL_TEXTURE_RECTANGLE, ?GL_TEXTURE_CUBE_MAP, ?GL_TEXTURE_CUBE_MAP_ARRAY specify one-, two-, or three-dimensional, one-dimensional array, two-dimensional array, rectangle, cube-mapped or cube-mapped array texturing, respectively. Pname accepts the same symbols as *gl:texParameterf/3* , with the same interpretations:

?GL_TEXTURE_MAG_FILTER: Returns the single-valued texture magnification filter, a symbolic constant. The initial value is ?GL_LINEAR.

?GL_TEXTURE_MIN_FILTER: Returns the single-valued texture minification filter, a symbolic constant. The initial value is ?GL_NEAREST_MIPMAP_LINEAR.

?GL_TEXTURE_MIN_LOD: Returns the single-valued texture minimum level-of-detail value. The initial value is -1000.

?GL_TEXTURE_MAX_LOD: Returns the single-valued texture maximum level-of-detail value. The initial value is 1000.

?GL_TEXTURE_BASE_LEVEL: Returns the single-valued base texture mipmap level. The initial value is 0.

?GL_TEXTURE_MAX_LEVEL: Returns the single-valued maximum texture mipmap array level. The initial value is 1000.

?GL_TEXTURE_SWIZZLE_R: Returns the red component swizzle. The initial value is ?GL_RED .

?GL_TEXTURE_SWIZZLE_G: Returns the green component swizzle. The initial value is ?GL_GREEN .

?GL_TEXTURE_SWIZZLE_B: Returns the blue component swizzle. The initial value is ?GL_BLUE .

?GL_TEXTURE_SWIZZLE_A: Returns the alpha component swizzle. The initial value is ?GL_ALPHA .

?GL_TEXTURE_SWIZZLE_RGBA: Returns the component swizzle for all channels in a single query.

?GL_TEXTURE_WRAP_S: Returns the single-valued wrapping function for texture coordinate s, a symbolic constant. The initial value is ?GL_REPEAT.

?GL_TEXTURE_WRAP_T: Returns the single-valued wrapping function for texture coordinate t, a symbolic constant. The initial value is ?GL_REPEAT.

?GL_TEXTURE_WRAP_R: Returns the single-valued wrapping function for texture coordinate r, a symbolic constant. The initial value is ?GL_REPEAT.

?GL_TEXTURE_BORDER_COLOR: Returns four integer or floating-point numbers that comprise the RGBA color of the texture border. Floating-point values are returned in the range [0 1]. Integer values are returned as a linear mapping of the internal floating-point representation such that 1.0 maps to the most positive representable integer and -1.0 maps to the most negative representable integer. The initial value is (0, 0, 0, 0).

?GL_TEXTURE_COMPARE_MODE: Returns a single-valued texture comparison mode, a symbolic constant. The initial value is ?GL_NONE. See *gl:texParameterf/3* .

?GL_TEXTURE_COMPARE_FUNC: Returns a single-valued texture comparison function, a symbolic constant. The initial value is ?GL_LEQUAL. See *gl:texParameterf/3* .

In addition to the parameters that may be set with *gl:texParameterf/3* , gl:getTexParameter accepts the following read-only parameters:

?GL_TEXTURE_IMMUTABLE_FORMAT: Returns non-zero if the texture has an immutable format. Textures become immutable if their storage is specified with *gl:texStorage1D/4* , *gl:texStorage2D/5* or *gl:texStorage3D/6* . The initial value is ?GL_FALSE .

See **external** documentation.

getTexParameteriv(Target, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Target = enum()

Pname = enum()

See *getTexParameterfv/2*

getTexLevelParameterfv(Target, Level, Pname) -> {float()}

Types:

Target = enum()

Level = integer()

Pname = enum()

Return texture parameter values for a specific level of detail

gl:getTexLevelParameter returns in Params texture parameter values for a specific level-of-detail value, specified as Level . Target defines the target texture, either ?GL_TEXTURE_1D, ?GL_TEXTURE_2D, ?GL_TEXTURE_3D, ?GL_PROXY_TEXTURE_1D , ?GL_PROXY_TEXTURE_2D, ?GL_PROXY_TEXTURE_3D, ?GL_TEXTURE_CUBE_MAP_POSITIVE_X , ?GL_TEXTURE_CUBE_MAP_NEGATIVE_X, ?GL_TEXTURE_CUBE_MAP_POSITIVE_Y, ?GL_TEXTURE_CUBE_MAP_NEGATIVE_Y , ?GL_TEXTURE_CUBE_MAP_POSITIVE_Z, ?GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, or ?GL_PROXY_TEXTURE_CUBE_MAP .

?GL_MAX_TEXTURE_SIZE, and ?GL_MAX_3D_TEXTURE_SIZE are not really descriptive enough. It has to report the largest square texture image that can be accommodated with mipmaps and borders, but a long skinny texture, or a texture without mipmaps and borders, may easily fit in texture memory. The proxy targets allow the user to more accurately query whether the GL can accommodate a texture of a given configuration. If the texture cannot be accommodated, the texture state variables, which may be queried with gl:getTexLevelParameter , are set to 0. If the texture can be accommodated, the texture state values will be set as they would be set for a non-proxy target.

Pname specifies the texture parameter whose value or values will be returned.

The accepted parameter names are as follows:

?GL_TEXTURE_WIDTH: Params returns a single value, the width of the texture image. This value includes the border of the texture image. The initial value is 0.

?GL_TEXTURE_HEIGHT: Params returns a single value, the height of the texture image. This value includes the border of the texture image. The initial value is 0.

?GL_TEXTURE_DEPTH: Params returns a single value, the depth of the texture image. This value includes the border of the texture image. The initial value is 0.

?GL_TEXTURE_INTERNAL_FORMAT: Params returns a single value, the internal format of the texture image.

?GL_TEXTURE_RED_TYPE,

?GL_TEXTURE_GREEN_TYPE,

?GL_TEXTURE_BLUE_TYPE,

?GL_TEXTURE_ALPHA_TYPE,

?GL_TEXTURE_DEPTH_TYPE: The data type used to store the component. The types ?GL_NONE , ?GL_SIGNED_NORMALIZED, ?GL_UNSIGNED_NORMALIZED, ?GL_FLOAT, ?GL_INT , and ?GL_UNSIGNED_INT may be returned to indicate signed normalized fixed-point, unsigned normalized fixed-point, floating-point, integer unnormalized, and unsigned integer unnormalized components, respectively.

?GL_TEXTURE_RED_SIZE,

?GL_TEXTURE_GREEN_SIZE,

?GL_TEXTURE_BLUE_SIZE,

?GL_TEXTURE_ALPHA_SIZE,

?GL_TEXTURE_DEPTH_SIZE: The internal storage resolution of an individual component. The resolution chosen by the GL will be a close match for the resolution requested by the user with the component argument of *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:copyTexImage1D/7* , and *gl:copyTexImage2D/8* . The initial value is 0.

?GL_TEXTURE_COMPRESSED: Params returns a single boolean value indicating if the texture image is stored in a compressed internal format. The initiali value is ?GL_FALSE .

?GL_TEXTURE_COMPRESSED_IMAGE_SIZE: Params returns a single integer value, the number of unsigned bytes of the compressed texture image that would be returned from *gl:getCompressedTexImage/3* .

See **external** documentation.

getTexLevelParameteriv(Target, Level, Pname) -> {integer()}

Types:

Target = enum()
Level = integer()
Pname = enum()

See *getTexLevelParameterfv/3*

texImage1D(Target, Level, InternalFormat, Width, Border, Format, Type, Pixels) -> ok

Types:

Target = enum()
Level = integer()
InternalFormat = integer()
Width = integer()
Border = integer()
Format = enum()
Type = enum()
Pixels = offset() | mem()

Specify a one-dimensional texture image

Texturing maps a portion of a specified texture image onto each graphical primitive for which texturing is enabled. To enable and disable one-dimensional texturing, call *gl:enable/1* and *gl:enable/1* with argument `?GL_TEXTURE_1D`.

Texture images are defined with `gl:texImage1D`. The arguments describe the parameters of the texture image, such as width, width of the border, level-of-detail number (see *gl:texParameterf/3*), and the internal resolution and format used to store the image. The last three arguments describe how the image is represented in memory.

If `Target` is `?GL_PROXY_TEXTURE_1D`, no data is read from `Data` , but all of the texture image state is recalculated, checked for consistency, and checked against the implementation's capabilities. If the implementation cannot handle a texture of the requested texture size, it sets all of the image state to 0, but does not generate an error (see *gl:getError/0*). To query for an entire mipmap array, use an image array level greater than or equal to 1.

If `Target` is `?GL_TEXTURE_1D`, data is read from `Data` as a sequence of signed or unsigned bytes, shorts, or longs, or single-precision floating-point values, depending on `Type` . These values are grouped into sets of one, two, three, or four values, depending on `Format` , to form elements. Each data byte is treated as eight 1-bit elements, with bit ordering determined by `?GL_UNPACK_LSB_FIRST` (see *gl:pixelStoref/2*).

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a texture image is specified, `Data` is treated as a byte offset into the buffer object's data store.

The first element corresponds to the left end of the texture array. Subsequent elements progress left-to-right through the remaining texels in the texture array. The final element corresponds to the right end of the texture array.

`Format` determines the composition of each element in `Data` . It can assume one of these symbolic values:

`?GL_RED`: Each element is a single red component. The GL converts it to floating point and assembles it into an RGBA element by attaching 0 for green and blue, and 1 for alpha. Each component is then multiplied by the signed scale factor `?GL_c_SCALE`, added to the signed bias `?GL_c_BIAS`, and clamped to the range [0,1].

?GL_RG: Each element is a single red/green double The GL converts it to floating point and assembles it into an RGBA element by attaching 0 for blue, and 1 for alpha. Each component is then multiplied by the signed scale factor ?GL_c_SCALE, added to the signed bias ?GL_c_BIAS, and clamped to the range [0,1].

?GL_RGB

?GL_BGR: Each element is an RGB triple. The GL converts it to floating point and assembles it into an RGBA element by attaching 1 for alpha. Each component is then multiplied by the signed scale factor ?GL_c_SCALE, added to the signed bias ?GL_c_BIAS, and clamped to the range [0,1].

?GL_RGBA

?GL_BGRA: Each element contains all four components. Each component is multiplied by the signed scale factor ?GL_c_SCALE, added to the signed bias ?GL_c_BIAS, and clamped to the range [0,1].

?GL_DEPTH_COMPONENT: Each element is a single depth value. The GL converts it to floating point, multiplies by the signed scale factor ?GL_DEPTH_SCALE, adds the signed bias ?GL_DEPTH_BIAS, and clamps to the range [0,1].

If an application wants to store the texture at a certain resolution or in a certain format, it can request the resolution and format with InternalFormat . The GL will choose an internal representation that closely approximates that requested by InternalFormat , but it may not match exactly. (The representations specified by ?GL_RED, ?GL_RG , ?GL_RGB and ?GL_RGBA must match exactly.)

InternalFormat may be one of the base internal formats shown in Table 1, below

InternalFormat may also be one of the sized internal formats shown in Table 2, below

Finally, InternalFormat may also be one of the generic or compressed compressed texture formats shown in Table 3 below

If the InternalFormat parameter is one of the generic compressed formats, ?GL_COMPRESSED_RED , ?GL_COMPRESSED_RG, ?GL_COMPRESSED_RGB, or ?GL_COMPRESSED_RGBA, the GL will replace the internal format with the symbolic constant for a specific internal format and compress the texture before storage. If no corresponding internal format is available, or the GL can not compress that image for any reason, the internal format is instead replaced with a corresponding base internal format.

If the InternalFormat parameter is ?GL_SRGB, ?GL_SRGB8, ?GL_SRGB_ALPHA or ?GL_SRGB8_ALPHA8, the texture is treated as if the red, green, or blue components are encoded in the sRGB color space. Any alpha component is left unchanged. The conversion from the sRGB encoded component c s to a linear component c l is:

c l={ c s/12.92if c s≤ 0.04045(c s+0.055/1.055) 2.4if c s> 0.04045

Assume c s is the sRGB component in the range [0,1].

Use the ?GL_PROXY_TEXTURE_1D target to try out a resolution and format. The implementation will update and recompute its best match for the requested storage resolution and format. To then query this state, call *gl:getTexLevelParameterfv/3* . If the texture cannot be accommodated, texture state is set to 0.

A one-component texture image uses only the red component of the RGBA color from Data . A two-component image uses the R and A values. A three-component image uses the R, G, and B values. A four-component image uses all of the RGBA components.

Image-based shadowing can be enabled by comparing texture r coordinates to depth texture values to generate a boolean result. See *gl:texParameterf/3* for details on texture comparison.

See **external** documentation.

texImage2D(Target, Level, InternalFormat, Width, Height, Border, Format, Type, Pixels) -> ok

Types:

Target = enum()

```
Level = integer()
InternalFormat = integer()
Width = integer()
Height = integer()
Border = integer()
Format = enum()
Type = enum()
Pixels = offset() | mem()
```

Specify a two-dimensional texture image

Texturing allows elements of an image array to be read by shaders.

To define texture images, call `gl:texImage2D`. The arguments describe the parameters of the texture image, such as height, width, width of the border, level-of-detail number (see *gl:texParameterf/3*), and number of color components provided. The last three arguments describe how the image is represented in memory.

If `Target` is `?GL_PROXY_TEXTURE_2D`, `?GL_PROXY_TEXTURE_1D_ARRAY`, `?GL_PROXY_TEXTURE_CUBE_MAP` , or `?GL_PROXY_TEXTURE_RECTANGLE`, no data is read from `Data` , but all of the texture image state is recalculated, checked for consistency, and checked against the implementation's capabilities. If the implementation cannot handle a texture of the requested texture size, it sets all of the image state to 0, but does not generate an error (see *gl:getError/0*). To query for an entire mipmap array, use an image array level greater than or equal to 1.

If `Target` is `?GL_TEXTURE_2D`, `?GL_TEXTURE_RECTANGLE` or one of the `?GL_TEXTURE_CUBE_MAP` targets, data is read from `Data` as a sequence of signed or unsigned bytes, shorts, or longs, or single-precision floating-point values, depending on `Type` . These values are grouped into sets of one, two, three, or four values, depending on `Format` , to form elements. Each data byte is treated as eight 1-bit elements, with bit ordering determined by `?GL_UNPACK_LSB_FIRST` (see *gl:pixelStoref/2*).

If `Target` is `?GL_TEXTURE_1D_ARRAY`, data is interpreted as an array of one-dimensional images.

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a texture image is specified, `Data` is treated as a byte offset into the buffer object's data store.

The first element corresponds to the lower left corner of the texture image. Subsequent elements progress left-to-right through the remaining texels in the lowest row of the texture image, and then in successively higher rows of the texture image. The final element corresponds to the upper right corner of the texture image.

`Format` determines the composition of each element in `Data` . It can assume one of these symbolic values:

`?GL_RED`: Each element is a single red component. The GL converts it to floating point and assembles it into an RGBA element by attaching 0 for green and blue, and 1 for alpha. Each component is then multiplied by the signed scale factor `?GL_c_SCALE`, added to the signed bias `?GL_c_BIAS`, and clamped to the range [0,1].

`?GL_RG`: Each element is a red/green double. The GL converts it to floating point and assembles it into an RGBA element by attaching 0 for blue, and 1 for alpha. Each component is then multiplied by the signed scale factor `?GL_c_SCALE`, added to the signed bias `?GL_c_BIAS`, and clamped to the range [0,1].

`?GL_RGB`

`?GL_BGR`: Each element is an RGB triple. The GL converts it to floating point and assembles it into an RGBA element by attaching 1 for alpha. Each component is then multiplied by the signed scale factor `?GL_c_SCALE`, added to the signed bias `?GL_c_BIAS`, and clamped to the range [0,1].

`?GL_RGBA`

`?GL_BGRA`: Each element contains all four components. Each component is multiplied by the signed scale factor `?GL_c_SCALE`, added to the signed bias `?GL_c_BIAS`, and clamped to the range [0,1].

?GL_DEPTH_COMPONENT: Each element is a single depth value. The GL converts it to floating point, multiplies by the signed scale factor ?GL_DEPTH_SCALE, adds the signed bias ?GL_DEPTH_BIAS, and clamps to the range [0,1].

?GL_DEPTH_STENCIL: Each element is a pair of depth and stencil values. The depth component of the pair is interpreted as in ?GL_DEPTH_COMPONENT. The stencil component is interpreted based on specified the depth + stencil internal format.

If an application wants to store the texture at a certain resolution or in a certain format, it can request the resolution and format with InternalFormat . The GL will choose an internal representation that closely approximates that requested by InternalFormat , but it may not match exactly. (The representations specified by ?GL_RED, ?GL_RG , ?GL_RGB, and ?GL_RGBA must match exactly.)

InternalFormat may be one of the base internal formats shown in Table 1, below

InternalFormat may also be one of the sized internal formats shown in Table 2, below

Finally, InternalFormat may also be one of the generic or compressed compressed texture formats shown in Table 3 below

If the InternalFormat parameter is one of the generic compressed formats, ?GL_COMPRESSED_RED , ?GL_COMPRESSED_RG, ?GL_COMPRESSED_RGB, or ?GL_COMPRESSED_RGBA, the GL will replace the internal format with the symbolic constant for a specific internal format and compress the texture before storage. If no corresponding internal format is available, or the GL can not compress that image for any reason, the internal format is instead replaced with a corresponding base internal format.

If the InternalFormat parameter is ?GL_SRGB, ?GL_SRGB8, ?GL_SRGB_ALPHA , or ?GL_SRGB8_ALPHA8, the texture is treated as if the red, green, or blue components are encoded in the sRGB color space. Any alpha component is left unchanged. The conversion from the sRGB encoded component c s to a linear component c l is:

c l={ c s/12.92if c s≤ 0.04045(c s+0.055/1.055) 2.4if c s> 0.04045

Assume c s is the sRGB component in the range [0,1].

Use the ?GL_PROXY_TEXTURE_2D, ?GL_PROXY_TEXTURE_1D_ARRAY, ?GL_PROXY_TEXTURE_RECTANGLE , or ?GL_PROXY_TEXTURE_CUBE_MAP target to try out a resolution and format. The implementation will update and recompute its best match for the requested storage resolution and format. To then query this state, call *gl:getTexLevelParameterfv/3* . If the texture cannot be accommodated, texture state is set to 0.

A one-component texture image uses only the red component of the RGBA color extracted from Data . A two-component image uses the R and G values. A three-component image uses the R, G, and B values. A four-component image uses all of the RGBA components.

Image-based shadowing can be enabled by comparing texture r coordinates to depth texture values to generate a boolean result. See *gl:texParameterf/3* for details on texture comparison.

See **external** documentation.

getTexImage(Target, Level, Format, Type, Pixels) -> ok

Types:

Target = enum()
Level = integer()
Format = enum()
Type = enum()
Pixels = mem()

Return a texture image

gl:getTexImage returns a texture image into Img . Target specifies whether the desired texture image is one specified by *gl:texImage1D/8* (?GL_TEXTURE_1D), *gl:texImage2D/9* (?GL_TEXTURE_1D_ARRAY, ?GL_TEXTURE_RECTANGLE, ?GL_TEXTURE_2D or any of ?GL_TEXTURE_CUBE_MAP_*), or *gl:texImage3D/10* (?GL_TEXTURE_2D_ARRAY , ?GL_TEXTURE_3D). Level specifies the level-of-detail number of the desired image. Format and Type specify the format and type of the desired image array. See the reference page for *gl:texImage1D/8* for a description of the acceptable values for the Format and Type parameters, respectively.

If a non-zero named buffer object is bound to the ?GL_PIXEL_PACK_BUFFER target (see *gl:bindBuffer/2*) while a texture image is requested, Img is treated as a byte offset into the buffer object's data store.

To understand the operation of gl:getTexImage, consider the selected internal four-component texture image to be an RGBA color buffer the size of the image. The semantics of gl:getTexImage are then identical to those of *gl:readPixels/7* , with the exception that no pixel transfer operations are performed, when called with the same Format and Type , with x and y set to 0, width set to the width of the texture image and height set to 1 for 1D images, or to the height of the texture image for 2D images.

If the selected texture image does not contain four components, the following mappings are applied. Single-component textures are treated as RGBA buffers with red set to the single-component value, green set to 0, blue set to 0, and alpha set to 1. Two-component textures are treated as RGBA buffers with red set to the value of component zero, alpha set to the value of component one, and green and blue set to 0. Finally, three-component textures are treated as RGBA buffers with red set to component zero, green set to component one, blue set to component two, and alpha set to 1.

To determine the required size of Img , use *gl:getTexLevelParameterfv/3* to determine the dimensions of the internal texture image, then scale the required number of pixels by the storage required for each pixel, based on Format and Type . Be sure to take the pixel storage parameters into account, especially ?GL_PACK_ALIGNMENT .

See **external** documentation.

genTextures(N) -> [integer()]

Types:

N = integer()

Generate texture names

gl:genTextures returns N texture names in Textures . There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to gl:genTextures.

The generated textures have no dimensionality; they assume the dimensionality of the texture target to which they are first bound (see *gl:bindTexture/2*).

Texture names returned by a call to gl:genTextures are not returned by subsequent calls, unless they are first deleted with *gl:deleteTextures/1* .

See **external** documentation.

deleteTextures(Textures) -> ok

Types:

Textures = [integer()]

Delete named textures

gl:deleteTextures deletes N textures named by the elements of the array Textures . After a texture is deleted, it has no contents or dimensionality, and its name is free for reuse (for example by *gl:genTextures/1*). If a texture that is currently bound is deleted, the binding reverts to 0 (the default texture).

gl:deleteTextures silently ignores 0's and names that do not correspond to existing textures.

See **external** documentation.

bindTexture(Target, Texture) -> ok

Types:

Target = enum()

Texture = integer()

Bind a named texture to a texturing target

`gl:bindTexture` lets you create or use a named texture. Calling `gl:bindTexture` with `Target` set to `?GL_TEXTURE_1D`, `?GL_TEXTURE_2D`, `?GL_TEXTURE_3D` , or `?GL_TEXTURE_1D_ARRAY`, `?GL_TEXTURE_2D_ARRAY`, `?GL_TEXTURE_RECTANGLE` , `?GL_TEXTURE_CUBE_MAP`, `?GL_TEXTURE_2D_MULTISAMPLE` or `?GL_TEXTURE_2D_MULTISAMPLE_ARRAY` and `Texture` set to the name of the new texture binds the texture name to the target. When a texture is bound to a target, the previous binding for that target is automatically broken.

Texture names are unsigned integers. The value zero is reserved to represent the default texture for each texture target. Texture names and the corresponding texture contents are local to the shared object space of the current GL rendering context; two rendering contexts share texture names only if they explicitly enable sharing between contexts through the appropriate GL windows interfaces functions.

You must use *gl:genTextures/1* to generate a set of new texture names.

When a texture is first bound, it assumes the specified target: A texture first bound to `?GL_TEXTURE_1D` becomes one-dimensional texture, a texture first bound to `?GL_TEXTURE_2D` becomes two-dimensional texture, a texture first bound to `?GL_TEXTURE_3D` becomes three-dimensional texture, a texture first bound to `?GL_TEXTURE_1D_ARRAY` becomes one-dimensional array texture, a texture first bound to `?GL_TEXTURE_2D_ARRAY` becomes two-dimensional arary texture, a texture first bound to `?GL_TEXTURE_RECTANGLE` becomes rectangle texture, a, texture first bound to `?GL_TEXTURE_CUBE_MAP` becomes a cube-mapped texture, a texture first bound to `?GL_TEXTURE_2D_MULTISAMPLE` becomes a two-dimensional multisampled texture, and a texture first bound to `?GL_TEXTURE_2D_MULTISAMPLE_ARRAY` becomes a two-dimensional multisampled array texture. The state of a one-dimensional texture immediately after it is first bound is equivalent to the state of the default `?GL_TEXTURE_1D` at GL initialization, and similarly for the other texture types.

While a texture is bound, GL operations on the target to which it is bound affect the bound texture, and queries of the target to which it is bound return state from the bound texture. In effect, the texture targets become aliases for the textures currently bound to them, and the texture name zero refers to the default textures that were bound to them at initialization.

A texture binding created with `gl:bindTexture` remains active until a different texture is bound to the same target, or until the bound texture is deleted with *gl:deleteTextures/1* .

Once created, a named texture may be re-bound to its same original target as often as needed. It is usually much faster to use `gl:bindTexture` to bind an existing named texture to one of the texture targets than it is to reload the texture image using *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* or another similar function.

See **external** documentation.

prioritizeTextures(Textures, Priorities) -> ok

Types:

Textures = [integer()]

Priorities = [clamp()]

Set texture residence priority

`gl:prioritizeTextures` assigns the `N` texture priorities given in `Priorities` to the `N` textures named in `Textures` .

The GL establishes a working set of textures that are resident in texture memory. These textures may be bound to a texture target much more efficiently than textures that are not resident. By specifying a priority for each texture, gl:prioritizeTextures allows applications to guide the GL implementation in determining which textures should be resident.

The priorities given in Priorities are clamped to the range [0 1] before they are assigned. 0 indicates the lowest priority; textures with priority 0 are least likely to be resident. 1 indicates the highest priority; textures with priority 1 are most likely to be resident. However, textures are not guaranteed to be resident until they are used.

gl:prioritizeTextures silently ignores attempts to prioritize texture 0 or any texture name that does not correspond to an existing texture.

gl:prioritizeTextures does not require that any of the textures named by Textures be bound to a texture target. *gl:texParameterf/3* may also be used to set a texture's priority, but only if the texture is currently bound. This is the only way to set the priority of a default texture.

See **external** documentation.

areTexturesResident(Textures) -> {0 | 1, Residences::[0 | 1]}

Types:

Textures = [integer()]

Determine if textures are loaded in texture memory

GL establishes a working set of textures that are resident in texture memory. These textures can be bound to a texture target much more efficiently than textures that are not resident.

gl:areTexturesResident queries the texture residence status of the N textures named by the elements of Textures . If all the named textures are resident, gl:areTexturesResident returns ?GL_TRUE, and the contents of Residences are undisturbed. If not all the named textures are resident, gl:areTexturesResident returns ?GL_FALSE, and detailed status is returned in the N elements of Residences . If an element of Residences is ?GL_TRUE, then the texture named by the corresponding element of Textures is resident.

The residence status of a single bound texture may also be queried by calling *gl:getTexParameterfv/2* with the target argument set to the target to which the texture is bound, and the pname argument set to ?GL_TEXTURE_RESIDENT. This is the only way that the residence status of a default texture can be queried.

See **external** documentation.

isTexture(Texture) -> 0 | 1

Types:

Texture = integer()

Determine if a name corresponds to a texture

gl:isTexture returns ?GL_TRUE if Texture is currently the name of a texture. If Texture is zero, or is a non-zero value that is not currently the name of a texture, or if an error occurs, gl:isTexture returns ?GL_FALSE.

A name returned by *gl:genTextures/1* , but not yet associated with a texture by calling *gl:bindTexture/2* , is not the name of a texture.

See **external** documentation.

texSubImage1D(Target, Level, Xoffset, Width, Format, Type, Pixels) -> ok

Types:

Target = enum()

Level = integer()

```
Xoffset = integer()
Width = integer()
Format = enum()
Type = enum()
Pixels = offset() | mem()
```

glTexSubImage

See **external** documentation.

texSubImage2D(Target, Level, Xoffset, Yoffset, Width, Height, Format, Type, Pixels) -> ok

Types:

```
Target = enum()
Level = integer()
Xoffset = integer()
Yoffset = integer()
Width = integer()
Height = integer()
Format = enum()
Type = enum()
Pixels = offset() | mem()
```

glTexSubImage

See **external** documentation.

copyTexImage1D(Target, Level, Internalformat, X, Y, Width, Border) -> ok

Types:

```
Target = enum()
Level = integer()
Internalformat = enum()
X = integer()
Y = integer()
Width = integer()
Border = integer()
```

Copy pixels into a 1D texture image

`gl:copyTexImage1D` defines a one-dimensional texture image with pixels from the current `?GL_READ_BUFFER`.

The screen-aligned pixel row with left corner at (x y) and with a length of width+2(border) defines the texture array at the mipmap level specified by `Level`. `Internalformat` specifies the internal format of the texture array.

The pixels in the row are processed exactly as if *gl:readPixels/7* had been called, but the process stops just before final conversion. At this point all pixel component values are clamped to the range [0 1] and then converted to the texture's internal format for storage in the texel array.

Pixel ordering is such that lower x screen coordinates correspond to lower texture coordinates.

If any of the pixels within the specified row of the current `?GL_READ_BUFFER` are outside the window associated with the current rendering context, then the values obtained for those pixels are undefined.

gl:copyTexImage1D defines a one-dimensional texture image with pixels from the current ?GL_READ_BUFFER.

When Internalformat is one of the sRGB types, the GL does not automatically convert the source pixels to the sRGB color space. In this case, the gl:pixelMap function can be used to accomplish the conversion.

See **external** documentation.

copyTexImage2D(Target, Level, Internalformat, X, Y, Width, Height, Border) -> ok

Types:

```
Target = enum()
Level = integer()
Internalformat = enum()
X = integer()
Y = integer()
Width = integer()
Height = integer()
Border = integer()
```

Copy pixels into a 2D texture image

gl:copyTexImage2D defines a two-dimensional texture image, or cube-map texture image with pixels from the current ?GL_READ_BUFFER.

The screen-aligned pixel rectangle with lower left corner at (X , Y) and with a width of width+2(border) and a height of height+2(border) defines the texture array at the mipmap level specified by Level . Internalformat specifies the internal format of the texture array.

The pixels in the rectangle are processed exactly as if *gl:readPixels/7* had been called, but the process stops just before final conversion. At this point all pixel component values are clamped to the range [0 1] and then converted to the texture's internal format for storage in the texel array.

Pixel ordering is such that lower x and y screen coordinates correspond to lower s and t texture coordinates.

If any of the pixels within the specified rectangle of the current ?GL_READ_BUFFER are outside the window associated with the current rendering context, then the values obtained for those pixels are undefined.

When Internalformat is one of the sRGB types, the GL does not automatically convert the source pixels to the sRGB color space. In this case, the gl:pixelMap function can be used to accomplish the conversion.

See **external** documentation.

copyTexSubImage1D(Target, Level, Xoffset, X, Y, Width) -> ok

Types:

```
Target = enum()
Level = integer()
Xoffset = integer()
X = integer()
Y = integer()
Width = integer()
```

Copy a one-dimensional texture subimage

gl:copyTexSubImage1D replaces a portion of a one-dimensional texture image with pixels from the current ? GL_READ_BUFFER (rather than from main memory, as is the case for *gl:texSubImage1D/7*).

The screen-aligned pixel row with left corner at (X , Y), and with length Width replaces the portion of the texture array with x indices Xoffset through xoffset +width-1, inclusive. The destination in the texture array may not include any texels outside the texture array as it was originally specified.

The pixels in the row are processed exactly as if *gl:readPixels/7* had been called, but the process stops just before final conversion. At this point, all pixel component values are clamped to the range [0 1] and then converted to the texture's internal format for storage in the texel array.

It is not an error to specify a subtexture with zero width, but such a specification has no effect. If any of the pixels within the specified row of the current ?GL_READ_BUFFER are outside the read window associated with the current rendering context, then the values obtained for those pixels are undefined.

No change is made to the internalformat, width, or border parameters of the specified texture array or to texel values outside the specified subregion.

See **external** documentation.

copyTexSubImage2D(Target, Level, Xoffset, Yoffset, X, Y, Width, Height) -> ok

Types:

```
Target = enum()
Level = integer()
Xoffset = integer()
Yoffset = integer()
X = integer()
Y = integer()
Width = integer()
Height = integer()
```

Copy a two-dimensional texture subimage

gl:copyTexSubImage2D replaces a rectangular portion of a two-dimensional texture image or cube-map texture image with pixels from the current ?GL_READ_BUFFER (rather than from main memory, as is the case for *gl:texSubImage1D/7*).

The screen-aligned pixel rectangle with lower left corner at (x y) and with width Width and height Height replaces the portion of the texture array with x indices Xoffset through xoffset+width-1, inclusive, and y indices Yoffset through yoffset+height -1, inclusive, at the mipmap level specified by Level .

The pixels in the rectangle are processed exactly as if *gl:readPixels/7* had been called, but the process stops just before final conversion. At this point, all pixel component values are clamped to the range [0 1] and then converted to the texture's internal format for storage in the texel array.

The destination rectangle in the texture array may not include any texels outside the texture array as it was originally specified. It is not an error to specify a subtexture with zero width or height, but such a specification has no effect.

If any of the pixels within the specified rectangle of the current ?GL_READ_BUFFER are outside the read window associated with the current rendering context, then the values obtained for those pixels are undefined.

No change is made to the internalformat, width, height, or border parameters of the specified texture array or to texel values outside the specified subregion.

See **external** documentation.

map1d(Target, U1, U2, Stride, Order, Points) -> ok

Types:

```
Target = enum()
```

```
U1 = float()
U2 = float()
Stride = integer()
Order = integer()
Points = binary()
```

glMap

See **external** documentation.

map1f(Target, U1, U2, Stride, Order, Points) -> ok

Types:

```
Target = enum()
U1 = float()
U2 = float()
Stride = integer()
Order = integer()
Points = binary()
```

glMap

See **external** documentation.

map2d(Target, U1, U2, Ustride, Uorder, V1, V2, Vstride, Vorder, Points) -> ok

Types:

```
Target = enum()
U1 = float()
U2 = float()
Ustride = integer()
Uorder = integer()
V1 = float()
V2 = float()
Vstride = integer()
Vorder = integer()
Points = binary()
```

glMap

See **external** documentation.

map2f(Target, U1, U2, Ustride, Uorder, V1, V2, Vstride, Vorder, Points) -> ok

Types:

```
Target = enum()
U1 = float()
U2 = float()
Ustride = integer()
Uorder = integer()
V1 = float()
```

```
V2 = float()
Vstride = integer()
Vorder = integer()
Points = binary()
```

glMap

See **external** documentation.

getMapdv(Target, Query, V) -> ok

Types:

```
Target = enum()
Query = enum()
V = mem()
```

Return evaluator parameters

gl:map1d/6 and *gl:map1d/6* define evaluators. `gl:getMap` returns evaluator parameters. `Target` chooses a map, `Query` selects a specific parameter, and `V` points to storage where the values will be returned.

The acceptable values for the `Target` parameter are described in the *gl:map1d/6* and *gl:map1d/6* reference pages.

`Query` can assume the following values:

`?GL_COEFF`: `V` returns the control points for the evaluator function. One-dimensional evaluators return order control points, and two-dimensional evaluators return uorder*vorder control points. Each control point consists of one, two, three, or four integer, single-precision floating-point, or double-precision floating-point values, depending on the type of the evaluator. The GL returns two-dimensional control points in row-major order, incrementing the uorder index quickly and the vorder index after each row. Integer values, when requested, are computed by rounding the internal floating-point values to the nearest integer values.

`?GL_ORDER`: `V` returns the order of the evaluator function. One-dimensional evaluators return a single value, order. The initial value is 1. Two-dimensional evaluators return two values, uorder and vorder. The initial value is 1,1.

`?GL_DOMAIN`: `V` returns the linear u and v mapping parameters. One-dimensional evaluators return two values, u1 and u2, as specified by *gl:map1d/6* . Two-dimensional evaluators return four values (u1, u2, v1, and v2) as specified by *gl:map1d/6* . Integer values, when requested, are computed by rounding the internal floating-point values to the nearest integer values.

See **external** documentation.

getMapfv(Target, Query, V) -> ok

Types:

```
Target = enum()
Query = enum()
V = mem()
```

See *getMapdv/3*

getMapiv(Target, Query, V) -> ok

Types:

```
Target = enum()
Query = enum()
V = mem()
```

See *getMapdv/3*

evalCoord1d(U) -> ok

Types:

U = float()

Evaluate enabled one- and two-dimensional maps

`gl:evalCoord1` evaluates enabled one-dimensional maps at argument `U` . `gl:evalCoord2` does the same for two-dimensional maps using two domain values, `U` and `V` . To define a map, call *gl:map1d/6* and *gl:map1d/6* ; to enable and disable it, call *gl:enable/1* and *gl:enable/1* .

When one of the `gl:evalCoord` commands is issued, all currently enabled maps of the indicated dimension are evaluated. Then, for each enabled map, it is as if the corresponding GL command had been issued with the computed value. That is, if `?GL_MAP1_INDEX` or `?GL_MAP2_INDEX` is enabled, a *gl:indexd/1* command is simulated. If `?GL_MAP1_COLOR_4` or `?GL_MAP2_COLOR_4` is enabled, a *gl:color3b/3* command is simulated. If `?GL_MAP1_NORMAL` or `?GL_MAP2_NORMAL` is enabled, a normal vector is produced, and if any of `?GL_MAP1_TEXTURE_COORD_1`, `?GL_MAP1_TEXTURE_COORD_2` , `?GL_MAP1_TEXTURE_COORD_3`, `?GL_MAP1_TEXTURE_COORD_4`, `?GL_MAP2_TEXTURE_COORD_1` , `?GL_MAP2_TEXTURE_COORD_2`, `?GL_MAP2_TEXTURE_COORD_3`, or `?GL_MAP2_TEXTURE_COORD_4` is enabled, then an appropriate *gl:texCoord1d/1* command is simulated.

For color, color index, normal, and texture coordinates the GL uses evaluated values instead of current values for those evaluations that are enabled, and current values otherwise, However, the evaluated values do not update the current values. Thus, if *gl:vertex2d/2* commands are interspersed with `gl:evalCoord` commands, the color, normal, and texture coordinates associated with the *gl:vertex2d/2* commands are not affected by the values generated by the `gl:evalCoord` commands, but only by the most recent *gl:color3b/3* , *gl:indexd/1* , *gl:normal3b/3* , and *gl:texCoord1d/1* commands.

No commands are issued for maps that are not enabled. If more than one texture evaluation is enabled for a particular dimension (for example, `?GL_MAP2_TEXTURE_COORD_1` and `?GL_MAP2_TEXTURE_COORD_2`), then only the evaluation of the map that produces the larger number of coordinates (in this case, `?GL_MAP2_TEXTURE_COORD_2`) is carried out. `?GL_MAP1_VERTEX_4` overrides `?GL_MAP1_VERTEX_3`, and `?GL_MAP2_VERTEX_4` overrides `?GL_MAP2_VERTEX_3` , in the same manner. If neither a three- nor a four-component vertex map is enabled for the specified dimension, the `gl:evalCoord` command is ignored.

If you have enabled automatic normal generation, by calling *gl:enable/1* with argument `?GL_AUTO_NORMAL`, `gl:evalCoord2` generates surface normals analytically, regardless of the contents or enabling of the `?GL_MAP2_NORMAL` map. Let

m=((∂ p)/(∂ u))*((∂ p)/(∂ v))

Then the generated normal n is n= m/(||m||)

If automatic normal generation is disabled, the corresponding normal map `?GL_MAP2_NORMAL` , if enabled, is used to produce a normal. If neither automatic normal generation nor a normal map is enabled, no normal is generated for `gl:evalCoord2` commands.

See **external** documentation.

evalCoord1f(U) -> ok

Types:

U = float()

See *evalCoord1d/1*

evalCoord1dv(U) -> ok

Types:

U = {U::float()}

Equivalent to *evalCoord1d(U)*.

evalCoord1fv(U) -> ok

Types:

U = {U::float()}

Equivalent to *evalCoord1f(U)*.

evalCoord2d(U, V) -> ok

Types:

U = float()

V = float()

See *evalCoord1d/1*

evalCoord2f(U, V) -> ok

Types:

U = float()

V = float()

See *evalCoord1d/1*

evalCoord2dv(U) -> ok

Types:

U = {U::float(), V::float()}

Equivalent to *evalCoord2d(U, V)*.

evalCoord2fv(U) -> ok

Types:

U = {U::float(), V::float()}

Equivalent to *evalCoord2f(U, V)*.

mapGrid1d(Un, U1, U2) -> ok

Types:

Un = integer()

U1 = float()

U2 = float()

Define a one- or two-dimensional mesh

`gl:mapGrid` and *gl:evalMesh1/3* are used together to efficiently generate and evaluate a series of evenly-spaced map domain values. *gl:evalMesh1/3* steps through the integer domain of a one- or two-dimensional grid, whose range is the domain of the evaluation maps specified by *gl:map1d/6* and *gl:map1d/6* .

`gl:mapGrid1` and `gl:mapGrid2` specify the linear grid mappings between the i (or i and j) integer grid coordinates, to the u (or u and v) floating-point evaluation map coordinates. See *gl:map1d/6* and *gl:map1d/6* for details of how u and v coordinates are evaluated.

`gl:mapGrid1` specifies a single linear mapping such that integer grid coordinate 0 maps exactly to `U1` , and integer grid coordinate `Un` maps exactly to `U2` . All other integer grid coordinates i are mapped so that

u= i(u2-u1)/un+u1

`gl:mapGrid2` specifies two such linear mappings. One maps integer grid coordinate i= 0 exactly to `U1` , and integer grid coordinate i= un exactly to `U2` . The other maps integer grid coordinate j= 0 exactly to `V1` , and integer grid coordinate j= vn exactly to `V2` . Other integer grid coordinates i and j are mapped such that

u= i(u2-u1)/un+u1

v= j(v2-v1)/vn+v1

The mappings specified by `gl:mapGrid` are used identically by *gl:evalMesh1/3* and *gl:evalPoint1/1* .

See **external** documentation.

`mapGrid1f(Un, U1, U2) -> ok`

Types:

```
Un = integer()
U1 = float()
U2 = float()
```

See *mapGrid1d/3*

`mapGrid2d(Un, U1, U2, Vn, V1, V2) -> ok`

Types:

```
Un = integer()
U1 = float()
U2 = float()
Vn = integer()
V1 = float()
V2 = float()
```

See *mapGrid1d/3*

`mapGrid2f(Un, U1, U2, Vn, V1, V2) -> ok`

Types:

```
Un = integer()
U1 = float()
U2 = float()
Vn = integer()
V1 = float()
V2 = float()
```

See *mapGrid1d/3*

`evalPoint1(I) -> ok`

Types:

I = integer()

Generate and evaluate a single point in a mesh

gl:mapGrid1d/3 and *gl:evalMesh1/3* are used in tandem to efficiently generate and evaluate a series of evenly spaced map domain values. `gl:evalPoint` can be used to evaluate a single grid point in the same gridspace that is traversed by *gl:evalMesh1/3* . Calling `gl:evalPoint1` is equivalent to calling glEvalCoord1(i.Δ u+u 1); where Δ u=(u 2-u 1)/n

and n, u 1, and u 2 are the arguments to the most recent *gl:mapGrid1d/3* command. The one absolute numeric requirement is that if i= n, then the value computed from i.Δ u+u 1 is exactly u 2.

In the two-dimensional case, `gl:evalPoint2`, let

Δ u=(u 2-u 1)/n

Δ v=(v 2-v 1)/m

where n, u 1, u 2, m, v 1, and v 2 are the arguments to the most recent *gl:mapGrid1d/3* command. Then the `gl:evalPoint2` command is equivalent to calling glEvalCoord2(i. Δ u+u 1, j.Δ v+v 1); The only absolute numeric requirements are that if i= n, then the value computed from i.Δ u+u 1 is exactly u 2, and if j= m, then the value computed from j.Δ v+v 1 is exactly v 2.

See **external** documentation.

evalPoint2(I, J) -> ok

Types:

I = integer()

J = integer()

See *evalPoint1/1*

evalMesh1(Mode, I1, I2) -> ok

Types:

Mode = enum()

I1 = integer()

I2 = integer()

Compute a one- or two-dimensional grid of points or lines

gl:mapGrid1d/3 and `gl:evalMesh` are used in tandem to efficiently generate and evaluate a series of evenly-spaced map domain values. `gl:evalMesh` steps through the integer domain of a one- or two-dimensional grid, whose range is the domain of the evaluation maps specified by *gl:map1d/6* and *gl:map1d/6* . `Mode` determines whether the resulting vertices are connected as points, lines, or filled polygons.

In the one-dimensional case, `gl:evalMesh1`, the mesh is generated as if the following code fragment were executed:

glBegin(`Type`); for (i = `I1` ; i <= `I2` ; i += 1) glEvalCoord1(i.Δ u+u 1); glEnd(); where

Δ u=(u 2-u 1)/n

and n, u 1, and u 2 are the arguments to the most recent *gl:mapGrid1d/3* command. `type` is `?GL_POINTS` if `Mode` is `?GL_POINT`, or `?GL_LINES` if `Mode` is `?GL_LINE`.

The one absolute numeric requirement is that if i= n, then the value computed from i. Δ u+u 1 is exactly u 2.

In the two-dimensional case, `gl:evalMesh2`, let .cp Δ u=(u 2-u 1)/n

Δ v=(v 2-v 1)/m

where n, u 1, u 2, m, v 1, and v 2 are the arguments to the most recent *gl:mapGrid1d/3* command. Then, if `Mode` is `?GL_FILL`, the `gl:evalMesh2` command is equivalent to:

for (j = `J1` ; j < `J2` ; j += 1) { glBegin(GL_QUAD_STRIP); for (i = `I1` ; i <= `I2` ; i += 1) { glEvalCoord2(i.Δ u+u 1, j.Δ v+v 1); glEvalCoord2(i.Δ u+u 1,(j+1).Δ v+v 1); } glEnd(); }

If `Mode` is `?GL_LINE`, then a call to `gl:evalMesh2` is equivalent to:

for (j = `J1` ; j <= `J2` ; j += 1) { glBegin(GL_LINE_STRIP); for (i = `I1` ; i <= `I2` ; i += 1) glEvalCoord2(i.Δ u+u 1, j.Δ v+v 1); glEnd(); } for (i = `I1` ; i <= `I2` ; i += 1) { glBegin(GL_LINE_STRIP); for (j = `J1` ; j <= `J1` ; j += 1) glEvalCoord2(i.Δ u+u 1, j. Δ v+v 1); glEnd(); }

And finally, if `Mode` is `?GL_POINT`, then a call to `gl:evalMesh2` is equivalent to:

glBegin(GL_POINTS); for (j = `J1` ; j <= `J2` ; j += 1) for (i = `I1` ; i <= `I2` ; i += 1) glEvalCoord2(i.Δ u+u 1, j.Δ v+v 1); glEnd();

In all three cases, the only absolute numeric requirements are that if i= n, then the value computed from i.Δ u +u 1 is exactly u 2, and if j= m, then the value computed from j.Δ v+v 1 is exactly v 2.

See **external** documentation.

`evalMesh2(Mode, I1, I2, J1, J2) -> ok`

Types:

```
Mode = enum()
I1 = integer()
I2 = integer()
J1 = integer()
J2 = integer()
```

See *evalMesh1/3*

`fogf(Pname, Param) -> ok`

Types:

```
Pname = enum()
Param = float()
```

Specify fog parameters

Fog is initially disabled. While enabled, fog affects rasterized geometry, bitmaps, and pixel blocks, but not buffer clear operations. To enable and disable fog, call *gl:enable/1* and *gl:enable/1* with argument `?GL_FOG`.

`gl:fog` assigns the value or values in `Params` to the fog parameter specified by `Pname` . The following values are accepted for `Pname` :

`?GL_FOG_MODE`: `Params` is a single integer or floating-point value that specifies the equation to be used to compute the fog blend factor, f. Three symbolic constants are accepted: `?GL_LINEAR`, `?GL_EXP`, and `?GL_EXP2`. The equations corresponding to these symbolic constants are defined below. The initial fog mode is `?GL_EXP`.

`?GL_FOG_DENSITY`: `Params` is a single integer or floating-point value that specifies density, the fog density used in both exponential fog equations. Only nonnegative densities are accepted. The initial fog density is 1.

`?GL_FOG_START`: `Params` is a single integer or floating-point value that specifies start, the near distance used in the linear fog equation. The initial near distance is 0.

`?GL_FOG_END`: `Params` is a single integer or floating-point value that specifies end, the far distance used in the linear fog equation. The initial far distance is 1.

?GL_FOG_INDEX: Params is a single integer or floating-point value that specifies i f, the fog color index. The initial fog index is 0.

?GL_FOG_COLOR: Params contains four integer or floating-point values that specify C f, the fog color. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. After conversion, all color components are clamped to the range [0 1]. The initial fog color is (0, 0, 0, 0).

?GL_FOG_COORD_SRC: Params contains either of the following symbolic constants: ?GL_FOG_COORD or ?GL_FRAGMENT_DEPTH. ?GL_FOG_COORD specifies that the current fog coordinate should be used as distance value in the fog color computation. ?GL_FRAGMENT_DEPTH specifies that the current fragment depth should be used as distance value in the fog computation.

Fog blends a fog color with each rasterized pixel fragment's post-texturing color using a blending factor f. Factor f is computed in one of three ways, depending on the fog mode. Let c be either the distance in eye coordinate from the origin (in the case that the ?GL_FOG_COORD_SRC is ?GL_FRAGMENT_DEPTH) or the current fog coordinate (in the case that ?GL_FOG_COORD_SRC is ?GL_FOG_COORD). The equation for ?GL_LINEAR fog is f=(end-c)/(end-start)

The equation for ?GL_EXP fog is f= e(-(density. c))

The equation for ?GL_EXP2 fog is f= e(-(density. c)) 2

Regardless of the fog mode, f is clamped to the range [0 1] after it is computed. Then, if the GL is in RGBA color mode, the fragment's red, green, and blue colors, represented by C r, are replaced by

(C r)"= f*C r+(1-f)*C f

Fog does not affect a fragment's alpha component.

In color index mode, the fragment's color index i r is replaced by

(i r)"= i r+(1-f)*i f

See **external** documentation.

fogi(Pname, Param) -> ok

Types:

Pname = enum()

Param = integer()

See *fogf/2*

fogfv(Pname, Params) -> ok

Types:

Pname = enum()

Params = {float()}

See *fogf/2*

fogiv(Pname, Params) -> ok

Types:

Pname = enum()

Params = {integer()}

See *fogf/2*

feedbackBuffer(Size, Type, Buffer) -> ok

Types:

Size = integer()

Type = enum()

Buffer = mem()

Controls feedback mode

The `gl:feedbackBuffer` function controls feedback. Feedback, like selection, is a GL mode. The mode is selected by calling *gl:renderMode/1* with `?GL_FEEDBACK`. When the GL is in feedback mode, no pixels are produced by rasterization. Instead, information about primitives that would have been rasterized is fed back to the application using the GL.

`gl:feedbackBuffer` has three arguments: `Buffer` is a pointer to an array of floating-point values into which feedback information is placed. `Size` indicates the size of the array. `Type` is a symbolic constant describing the information that is fed back for each vertex. `gl:feedbackBuffer` must be issued before feedback mode is enabled (by calling *gl:renderMode/1* with argument `?GL_FEEDBACK`). Setting `?GL_FEEDBACK` without establishing the feedback buffer, or calling `gl:feedbackBuffer` while the GL is in feedback mode, is an error.

When *gl:renderMode/1* is called while in feedback mode, it returns the number of entries placed in the feedback array and resets the feedback array pointer to the base of the feedback buffer. The returned value never exceeds `Size` . If the feedback data required more room than was available in `Buffer` , *gl:renderMode/1* returns a negative value. To take the GL out of feedback mode, call *gl:renderMode/1* with a parameter value other than `?GL_FEEDBACK`.

While in feedback mode, each primitive, bitmap, or pixel rectangle that would be rasterized generates a block of values that are copied into the feedback array. If doing so would cause the number of entries to exceed the maximum, the block is partially written so as to fill the array (if there is any room left at all), and an overflow flag is set. Each block begins with a code indicating the primitive type, followed by values that describe the primitive's vertices and associated data. Entries are also written for bitmaps and pixel rectangles. Feedback occurs after polygon culling and *gl:polygonMode/2* interpretation of polygons has taken place, so polygons that are culled are not returned in the feedback buffer. It can also occur after polygons with more than three edges are broken up into triangles, if the GL implementation renders polygons by performing this decomposition.

The *gl:passThrough/1* command can be used to insert a marker into the feedback buffer. See *gl:passThrough/1* .

Following is the grammar for the blocks of values written into the feedback buffer. Each primitive is indicated with a unique identifying value followed by some number of vertices. Polygon entries include an integer value indicating how many vertices follow. A vertex is fed back as some number of floating-point values, as determined by `Type` . Colors are fed back as four values in RGBA mode and one value in color index mode.

feedbackList feedbackItem feedbackList | feedbackItem

feedbackItem point | lineSegment | polygon | bitmap | pixelRectangle | passThru

point `?GL_POINT_TOKEN` vertex

lineSegment `?GL_LINE_TOKEN` vertex vertex | `?GL_LINE_RESET_TOKEN` vertex vertex

polygon `?GL_POLYGON_TOKEN` n polySpec

polySpec polySpec vertex | vertex vertex vertex

bitmap `?GL_BITMAP_TOKEN` vertex

pixelRectangle `?GL_DRAW_PIXEL_TOKEN` vertex | `?GL_COPY_PIXEL_TOKEN` vertex

passThru `?GL_PASS_THROUGH_TOKEN` value

vertex 2d | 3d | 3dColor | 3dColorTexture | 4dColorTexture

2d value value

3d value value value

3dColor value value value color

3dColorTexture value value value color tex

4dColorTexture value value value value color tex

color rgba | index

rgba value value value value

index value

tex value value value value

`value` is a floating-point number, and `n` is a floating-point integer giving the number of vertices in the polygon. `?GL_POINT_TOKEN`, `?GL_LINE_TOKEN`, `?GL_LINE_RESET_TOKEN` , `?GL_POLYGON_TOKEN`, `?GL_BITMAP_TOKEN`, `?GL_DRAW_PIXEL_TOKEN`, `?GL_COPY_PIXEL_TOKEN` and `?GL_PASS_THROUGH_TOKEN` are symbolic floating-point constants. `?GL_LINE_RESET_TOKEN` is returned whenever the line stipple pattern is reset. The data returned as a vertex depends on the feedback `Type` .

The following table gives the correspondence between `Type` and the number of values per vertex. `k` is 1 in color index mode and 4 in RGBA mode.

Type	Coordinates	Color	Texture	Total Number of Values
`?GL_2D`	`x`, `y`			2
`?GL_3D`	`x`, `y`, `z`			3
`?GL_3D_COLOR`	`x`, `y`, `z`	k		3+k
`?GL_3D_COLOR_TEXTURE`	`x`, `y`, `z`	k	4	7+k
`?GL_4D_COLOR_TEXTURE`	`x`, `y`, `z`, `w`	k	4	8+k

Feedback vertex coordinates are in window coordinates, except `w`, which is in clip coordinates. Feedback colors are lighted, if lighting is enabled. Feedback texture coordinates are generated, if texture coordinate generation is enabled. They are always transformed by the texture matrix.

See **external** documentation.

passThrough(Token) -> ok

Types:

Token = float()

Place a marker in the feedback buffer

Feedback is a GL render mode. The mode is selected by calling *gl:renderMode/1* with `?GL_FEEDBACK`. When the GL is in feedback mode, no pixels are produced by rasterization. Instead, information about primitives that would have been rasterized is fed back to the application using the GL. See the *gl:feedbackBuffer/3* reference page for a description of the feedback buffer and the values in it.

`gl:passThrough` inserts a user-defined marker in the feedback buffer when it is executed in feedback mode. `Token` is returned as if it were a primitive; it is indicated with its own unique identifying value: `?GL_PASS_THROUGH_TOKEN`. The order of `gl:passThrough` commands with respect to the specification of graphics primitives is maintained.

See **external** documentation.

selectBuffer(Size, Buffer) -> ok

Types:

Size = integer()

Buffer = mem()

Establish a buffer for selection mode values

`gl:selectBuffer` has two arguments: `Buffer` is a pointer to an array of unsigned integers, and `Size` indicates the size of the array. `Buffer` returns values from the name stack (see *gl:initNames/0* , *gl:loadName/1* , *gl:pushName/1*) when the rendering mode is `?GL_SELECT` (see *gl:renderMode/1*). `gl:selectBuffer` must be issued before selection mode is enabled, and it must not be issued while the rendering mode is `?GL_SELECT`.

A programmer can use selection to determine which primitives are drawn into some region of a window. The region is defined by the current modelview and perspective matrices.

In selection mode, no pixel fragments are produced from rasterization. Instead, if a primitive or a raster position intersects the clipping volume defined by the viewing frustum and the user-defined clipping planes, this primitive causes a selection hit. (With polygons, no hit occurs if the polygon is culled.) When a change is made to the name stack, or when *gl:renderMode/1* is called, a hit record is copied to `Buffer` if any hits have occurred since the last such event (name stack change or *gl:renderMode/1* call). The hit record consists of the number of names in the name stack at the time of the event, followed by the minimum and maximum depth values of all vertices that hit since the previous event, followed by the name stack contents, bottom name first.

Depth values (which are in the range [0,1]) are multiplied by 2 32-1, before being placed in the hit record.

An internal index into `Buffer` is reset to 0 whenever selection mode is entered. Each time a hit record is copied into `Buffer` , the index is incremented to point to the cell just past the end of the block of names(emthat is, to the next available cell If the hit record is larger than the number of remaining locations in `Buffer` , as much data as can fit is copied, and the overflow flag is set. If the name stack is empty when a hit record is copied, that record consists of 0 followed by the minimum and maximum depth values.

To exit selection mode, call *gl:renderMode/1* with an argument other than `?GL_SELECT` . Whenever *gl:renderMode/1* is called while the render mode is `?GL_SELECT`, it returns the number of hit records copied to `Buffer` , resets the overflow flag and the selection buffer pointer, and initializes the name stack to be empty. If the overflow bit was set when *gl:renderMode/1* was called, a negative hit record count is returned.

See **external** documentation.

`initNames() -> ok`

Initialize the name stack

The name stack is used during selection mode to allow sets of rendering commands to be uniquely identified. It consists of an ordered set of unsigned integers. `gl:initNames` causes the name stack to be initialized to its default empty state.

The name stack is always empty while the render mode is not `?GL_SELECT`. Calls to `gl:initNames` while the render mode is not `?GL_SELECT` are ignored.

See **external** documentation.

`loadName(Name) -> ok`

Types:

```
Name = integer()
```

Load a name onto the name stack

The name stack is used during selection mode to allow sets of rendering commands to be uniquely identified. It consists of an ordered set of unsigned integers and is initially empty.

`gl:loadName` causes `Name` to replace the value on the top of the name stack.

The name stack is always empty while the render mode is not `?GL_SELECT`. Calls to `gl:loadName` while the render mode is not `?GL_SELECT` are ignored.

See **external** documentation.

pushName(Name) -> ok

Types:

Name = integer()

Push and pop the name stack

The name stack is used during selection mode to allow sets of rendering commands to be uniquely identified. It consists of an ordered set of unsigned integers and is initially empty.

`gl:pushName` causes `Name` to be pushed onto the name stack. *gl:pushName/1* pops one name off the top of the stack.

The maximum name stack depth is implementation-dependent; call `?GL_MAX_NAME_STACK_DEPTH` to find out the value for a particular implementation. It is an error to push a name onto a full stack or to pop a name off an empty stack. It is also an error to manipulate the name stack between the execution of *gl:'begin'/1* and the corresponding execution of *gl:'begin'/1* . In any of these cases, the error flag is set and no other change is made to GL state.

The name stack is always empty while the render mode is not `?GL_SELECT`. Calls to `gl:pushName` or *gl:pushName/1* while the render mode is not `?GL_SELECT` are ignored.

See **external** documentation.

popName() -> ok

See *pushName/1*

blendColor(Red, Green, Blue, Alpha) -> ok

Types:

Red = clamp()
Green = clamp()
Blue = clamp()
Alpha = clamp()

Set the blend color

The `?GL_BLEND_COLOR` may be used to calculate the source and destination blending factors. The color components are clamped to the range [0 1] before being stored. See *gl:blendFunc/2* for a complete description of the blending operations. Initially the `?GL_BLEND_COLOR` is set to (0, 0, 0, 0).

See **external** documentation.

blendEquation(Mode) -> ok

Types:

Mode = enum()

Specify the equation used for both the RGB blend equation and the Alpha blend equation

The blend equations determine how a new pixel (the "source" color) is combined with a pixel already in the framebuffer (the "destination" color). This function sets both the RGB blend equation and the alpha blend equation to a single equation. `gl:blendEquationi` specifies the blend equation for a single draw buffer whereas `gl:blendEquation` sets the blend equation for all draw buffers.

These equations use the source and destination blend factors specified by either *gl:blendFunc/2* or *gl:blendFuncSeparate/4* . See *gl:blendFunc/2* or *gl:blendFuncSeparate/4* for a description of the various blend factors.

In the equations that follow, source and destination color components are referred to as (R s G s B s A s) and (R d G d B d A d), respectively. The result color is referred to as (R r G r B r A r). The source and destination blend factors are denoted (s R s G s B s A) and (d R d G d B d A), respectively. For these equations all color components are understood to have values in the range [0 1].

`Mode` `RGB ComponentsAlpha Component`
`?GL_FUNC_ADD` Rr= R s s R+R d d R Gr= G s s G+G d d G Br= B s s B+B d d B Ar= A s s A+A d d A
`?GL_FUNC_SUBTRACT` Rr= R s s R-R d d R Gr= G s s G-G d d G Br= B s s B-B d d B Ar= A s s A-A d d A
`?GL_FUNC_REVERSE_SUBTRACT` Rr= R d d R-R s s R Gr= G d d G-G s s G Br= B d d B-B s s B Ar= A d d A-A s s A
`?GL_MIN` Rr= min(R s R d) Gr= min(G s G d) Br= min(B s B d) Ar= min(A s A d)
`?GL_MAX` Rr= max(R s R d) Gr= max(G s G d) Br= max(B s B d) Ar= max(A s A d)

The results of these equations are clamped to the range [0 1].

The `?GL_MIN` and `?GL_MAX` equations are useful for applications that analyze image data (image thresholding against a constant color, for example). The `?GL_FUNC_ADD` equation is useful for antialiasing and transparency, among other things.

Initially, both the RGB blend equation and the alpha blend equation are set to `?GL_FUNC_ADD` .

See **external** documentation.

`drawRangeElements(Mode, Start, End, Count, Type, Indices) -> ok`

Types:

```
Mode = enum()
Start = integer()
End = integer()
Count = integer()
Type = enum()
Indices = offset() | mem()
```

Render primitives from array data

`gl:drawRangeElements` is a restricted form of *gl:drawElements/4* . `Mode` , `Start` , `End` , and `Count` match the corresponding arguments to *gl:drawElements/4* , with the additional constraint that all values in the arrays `Count` must lie between `Start` and `End` , inclusive.

Implementations denote recommended maximum amounts of vertex and index data, which may be queried by calling *gl:getBooleanv/1* with argument `?GL_MAX_ELEMENTS_VERTICES` and `?GL_MAX_ELEMENTS_INDICES` . If end-start+1 is greater than the value of `?GL_MAX_ELEMENTS_VERTICES`, or if `Count` is greater than the value of `?GL_MAX_ELEMENTS_INDICES`, then the call may operate at reduced performance. There is no requirement that all vertices in the range [start end] be referenced. However, the implementation may partially process unused vertices, reducing performance from what could be achieved with an optimal index set.

When `gl:drawRangeElements` is called, it uses `Count` sequential elements from an enabled array, starting at `Start` to construct a sequence of geometric primitives. `Mode` specifies what kind of primitives are constructed, and how the array elements construct these primitives. If more than one array is enabled, each is used.

Vertex attributes that are modified by `gl:drawRangeElements` have an unspecified value after `gl:drawRangeElements` returns. Attributes that aren't modified maintain their previous values.

See **external** documentation.

texImage3D(Target, Level, InternalFormat, Width, Height, Depth, Border, Format, Type, Pixels) -> ok

Types:

Target = enum()
Level = integer()
InternalFormat = integer()
Width = integer()
Height = integer()
Depth = integer()
Border = integer()
Format = enum()
Type = enum()
Pixels = offset() | mem()

Specify a three-dimensional texture image

Texturing maps a portion of a specified texture image onto each graphical primitive for which texturing is enabled. To enable and disable three-dimensional texturing, call *gl:enable/1* and *gl:enable/1* with argument `?GL_TEXTURE_3D`.

To define texture images, call `gl:texImage3D`. The arguments describe the parameters of the texture image, such as height, width, depth, width of the border, level-of-detail number (see *gl:texParameterf/3*), and number of color components provided. The last three arguments describe how the image is represented in memory.

If `Target` is `?GL_PROXY_TEXTURE_3D`, no data is read from `Data` , but all of the texture image state is recalculated, checked for consistency, and checked against the implementation's capabilities. If the implementation cannot handle a texture of the requested texture size, it sets all of the image state to 0, but does not generate an error (see *gl:getError/0*). To query for an entire mipmap array, use an image array level greater than or equal to 1.

If `Target` is `?GL_TEXTURE_3D`, data is read from `Data` as a sequence of signed or unsigned bytes, shorts, or longs, or single-precision floating-point values, depending on `Type` . These values are grouped into sets of one, two, three, or four values, depending on `Format` , to form elements. Each data byte is treated as eight 1-bit elements, with bit ordering determined by `?GL_UNPACK_LSB_FIRST` (see *gl:pixelStoref/2*).

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a texture image is specified, `Data` is treated as a byte offset into the buffer object's data store.

The first element corresponds to the lower left corner of the texture image. Subsequent elements progress left-to-right through the remaining texels in the lowest row of the texture image, and then in successively higher rows of the texture image. The final element corresponds to the upper right corner of the texture image.

`Format` determines the composition of each element in `Data` . It can assume one of these symbolic values:

`?GL_RED`: Each element is a single red component. The GL converts it to floating point and assembles it into an RGBA element by attaching 0 for green and blue, and 1 for alpha. Each component is then multiplied by the signed scale factor `?GL_c_SCALE`, added to the signed bias `?GL_c_BIAS`, and clamped to the range [0,1].

`?GL_RG`: Each element is a red and green pair. The GL converts each to floating point and assembles it into an RGBA element by attaching 0 for blue, and 1 for alpha. Each component is then multiplied by the signed scale factor `?GL_c_SCALE`, added to the signed bias `?GL_c_BIAS`, and clamped to the range [0,1].

`?GL_RGB`

`?GL_BGR`: Each element is an RGB triple. The GL converts it to floating point and assembles it into an RGBA element by attaching 1 for alpha. Each component is then multiplied by the signed scale factor `?GL_c_SCALE`, added to the signed bias `?GL_c_BIAS`, and clamped to the range [0,1].

`?GL_RGBA`

?GL_BGRA: Each element contains all four components. Each component is multiplied by the signed scale factor ?GL_c_SCALE, added to the signed bias ?GL_c_BIAS, and clamped to the range [0,1].

If an application wants to store the texture at a certain resolution or in a certain format, it can request the resolution and format with InternalFormat . The GL will choose an internal representation that closely approximates that requested by InternalFormat , but it may not match exactly. (The representations specified by ?GL_RED, ?GL_RG , ?GL_RGB, and ?GL_RGBA must match exactly.)

InternalFormat may be one of the base internal formats shown in Table 1, below

InternalFormat may also be one of the sized internal formats shown in Table 2, below

Finally, InternalFormat may also be one of the generic or compressed compressed texture formats shown in Table 3 below

If the InternalFormat parameter is one of the generic compressed formats, ?GL_COMPRESSED_RED , ?GL_COMPRESSED_RG, ?GL_COMPRESSED_RGB, or ?GL_COMPRESSED_RGBA, the GL will replace the internal format with the symbolic constant for a specific internal format and compress the texture before storage. If no corresponding internal format is available, or the GL can not compress that image for any reason, the internal format is instead replaced with a corresponding base internal format.

If the InternalFormat parameter is ?GL_SRGB, ?GL_SRGB8, ?GL_SRGB_ALPHA , or ?GL_SRGB8_ALPHA8, the texture is treated as if the red, green, blue, or luminance components are encoded in the sRGB color space. Any alpha component is left unchanged. The conversion from the sRGB encoded component c s to a linear component c l is:

c l={ c s/12.92if c s≤ 0.04045(c s+0.055/1.055) 2.4if c s> 0.04045

Assume c s is the sRGB component in the range [0,1].

Use the ?GL_PROXY_TEXTURE_3D target to try out a resolution and format. The implementation will update and recompute its best match for the requested storage resolution and format. To then query this state, call *gl:getTexLevelParameterfv/3* . If the texture cannot be accommodated, texture state is set to 0.

A one-component texture image uses only the red component of the RGBA color extracted from Data . A two-component image uses the R and A values. A three-component image uses the R, G, and B values. A four-component image uses all of the RGBA components.

See **external** documentation.

texSubImage3D(Target, Level, Xoffset, Yoffset, Zoffset, Width, Height, Depth, Format, Type, Pixels) -> ok

Types:

```
Target = enum()
Level = integer()
Xoffset = integer()
Yoffset = integer()
Zoffset = integer()
Width = integer()
Height = integer()
Depth = integer()
Format = enum()
Type = enum()
Pixels = offset() | mem()
```

glTexSubImage

See **external** documentation.

copyTexSubImage3D(Target, Level, Xoffset, Yoffset, Zoffset, X, Y, Width, Height) -> ok

Types:

Target = enum()
Level = integer()
Xoffset = integer()
Yoffset = integer()
Zoffset = integer()
X = integer()
Y = integer()
Width = integer()
Height = integer()

Copy a three-dimensional texture subimage

`gl:copyTexSubImage3D` replaces a rectangular portion of a three-dimensional texture image with pixels from the current `?GL_READ_BUFFER` (rather than from main memory, as is the case for *gl:texSubImage1D/7*).

The screen-aligned pixel rectangle with lower left corner at (`X` , `Y`) and with width `Width` and height `Height` replaces the portion of the texture array with x indices `Xoffset` through xoffset+width-1, inclusive, and y indices `Yoffset` through yoffset+height-1, inclusive, at z index `Zoffset` and at the mipmap level specified by `Level` .

The pixels in the rectangle are processed exactly as if *gl:readPixels/7* had been called, but the process stops just before final conversion. At this point, all pixel component values are clamped to the range [0 1] and then converted to the texture's internal format for storage in the texel array.

The destination rectangle in the texture array may not include any texels outside the texture array as it was originally specified. It is not an error to specify a subtexture with zero width or height, but such a specification has no effect.

If any of the pixels within the specified rectangle of the current `?GL_READ_BUFFER` are outside the read window associated with the current rendering context, then the values obtained for those pixels are undefined.

No change is made to the `internalformat`, `width`, `height`, `depth`, or `border` parameters of the specified texture array or to texel values outside the specified subregion.

See **external** documentation.

colorTable(Target, Internalformat, Width, Format, Type, Table) -> ok

Types:

Target = enum()
Internalformat = enum()
Width = integer()
Format = enum()
Type = enum()
Table = offset() | mem()

Define a color lookup table

`gl:colorTable` may be used in two ways: to test the actual size and color resolution of a lookup table given a particular set of parameters, or to load the contents of a color lookup table. Use the targets `?GL_PROXY_*` for the first case and the other targets for the second case.

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a color table is specified, `Data` is treated as a byte offset into the buffer object's data store.

If Target is ?GL_COLOR_TABLE, ?GL_POST_CONVOLUTION_COLOR_TABLE, or ?GL_POST_COLOR_MATRIX_COLOR_TABLE , gl:colorTable builds a color lookup table from an array of pixels. The pixel array specified by Width , Format , Type , and Data is extracted from memory and processed just as if *gl:drawPixels/5* were called, but processing stops after the final expansion to RGBA is completed.

The four scale parameters and the four bias parameters that are defined for the table are then used to scale and bias the R, G, B, and A components of each pixel. (Use gl:colorTableParameter to set these scale and bias parameters.)

Next, the R, G, B, and A values are clamped to the range [0 1]. Each pixel is then converted to the internal format specified by Internalformat . This conversion simply maps the component values of the pixel (R, G, B, and A) to the values included in the internal format (red, green, blue, alpha, luminance, and intensity). The mapping is as follows:

Internal FormatRedGreen BlueAlphaLuminanceIntensity
?GL_ALPHA A
?GL_LUMINANCE R
?GL_LUMINANCE_ALPHA A R
?GL_INTENSITY R
?GL_RGB R G B
?GL_RGBA R G B A

Finally, the red, green, blue, alpha, luminance, and/or intensity components of the resulting pixels are stored in the color table. They form a one-dimensional table with indices in the range [0 width-1].

If Target is ?GL_PROXY_*, gl:colorTable recomputes and stores the values of the proxy color table's state variables ?GL_COLOR_TABLE_FORMAT, ?GL_COLOR_TABLE_WIDTH , ?GL_COLOR_TABLE_RED_SIZE, ?GL_COLOR_TABLE_GREEN_SIZE, ?GL_COLOR_TABLE_BLUE_SIZE , ?GL_COLOR_TABLE_ALPHA_SIZE, ?GL_COLOR_TABLE_LUMINANCE_SIZE, and ?GL_COLOR_TABLE_INTENSITY_SIZE . There is no effect on the image or state of any actual color table. If the specified color table is too large to be supported, then all the proxy state variables listed above are set to zero. Otherwise, the color table could be supported by gl:colorTable using the corresponding non-proxy target, and the proxy state variables are set as if that target were being defined.

The proxy state variables can be retrieved by calling *gl:getColorTableParameterfv/2* with a target of ?GL_PROXY_*. This allows the application to decide if a particular gl:colorTable command would succeed, and to determine what the resulting color table attributes would be.

If a color table is enabled, and its width is non-zero, then its contents are used to replace a subset of the components of each RGBA pixel group, based on the internal format of the table.

Each pixel group has color components (R, G, B, A) that are in the range [0.0 1.0]. The color components are rescaled to the size of the color lookup table to form an index. Then a subset of the components based on the internal format of the table are replaced by the table entry selected by that index. If the color components and contents of the table are represented as follows:

RepresentationMeaning
r Table index computed from R
g Table index computed from G
b Table index computed from B
a Table index computed from A
L[i] Luminance value at table index i
I[i] Intensity value at table index i
R[i] Red value at table index i
G[i] Green value at table index i
B[i] Blue value at table index i
A[i] Alpha value at table index i

then the result of color table lookup is as follows:

Resulting Texture Components
Table Internal Format RGBA
?GL_ALPHARGBA[a]
?GL_LUMINANCEL[r]L[g]L[b]At
?GL_LUMINANCE_ALPHA L[r]L[g]L[b]A[a]
?GL_INTENSITY I[r]I[g]I[b]I[a]
?GL_RGBR[r] G[g]B[b]A
?GL_RGBAR[r] G[g]B[b]A[a]

When ?GL_COLOR_TABLE is enabled, the colors resulting from the pixel map operation (if it is enabled) are mapped by the color lookup table before being passed to the convolution operation. The colors resulting from the convolution operation are modified by the post convolution color lookup table when ?GL_POST_CONVOLUTION_COLOR_TABLE is enabled. These modified colors are then sent to the color matrix operation. Finally, if ?GL_POST_COLOR_MATRIX_COLOR_TABLE is enabled, the colors resulting from the color matrix operation are mapped by the post color matrix color lookup table before being used by the histogram operation.

See **external** documentation.

colorTableParameterfv(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {float(), float(), float(), float()}

Set color lookup table parameters

gl:colorTableParameter is used to specify the scale factors and bias terms applied to color components when they are loaded into a color table. Target indicates which color table the scale and bias terms apply to; it must be set to ?GL_COLOR_TABLE, ?GL_POST_CONVOLUTION_COLOR_TABLE , or ?GL_POST_COLOR_MATRIX_COLOR_TABLE.

Pname must be ?GL_COLOR_TABLE_SCALE to set the scale factors. In this case, Params points to an array of four values, which are the scale factors for red, green, blue, and alpha, in that order.

Pname must be ?GL_COLOR_TABLE_BIAS to set the bias terms. In this case, Params points to an array of four values, which are the bias terms for red, green, blue, and alpha, in that order.

The color tables themselves are specified by calling *gl:colorTable/6* .

See **external** documentation.

colorTableParameteriv(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {integer(), integer(), integer(), integer()}

See *colorTableParameterfv/3*

copyColorTable(Target, Internalformat, X, Y, Width) -> ok

Types:

Target = enum()
Internalformat = enum()
X = integer()

Y = integer()
Width = integer()

Copy pixels into a color table

`gl:copyColorTable` loads a color table with pixels from the current `?GL_READ_BUFFER` (rather than from main memory, as is the case for *gl:colorTable/6*).

The screen-aligned pixel rectangle with lower-left corner at (`X` , `Y`) having width `Width` and height 1 is loaded into the color table. If any pixels within this region are outside the window that is associated with the GL context, the values obtained for those pixels are undefined.

The pixels in the rectangle are processed just as if *gl:readPixels/7* were called, with `Internalformat` set to RGBA, but processing stops after the final conversion to RGBA.

The four scale parameters and the four bias parameters that are defined for the table are then used to scale and bias the R, G, B, and A components of each pixel. The scale and bias parameters are set by calling *gl:colorTableParameterfv/3* .

Next, the R, G, B, and A values are clamped to the range [0 1]. Each pixel is then converted to the internal format specified by `Internalformat` . This conversion simply maps the component values of the pixel (R, G, B, and A) to the values included in the internal format (red, green, blue, alpha, luminance, and intensity). The mapping is as follows:

`Internal FormatRedGreen BlueAlphaLuminanceIntensity`
`?GL_ALPHA` A
`?GL_LUMINANCE` R
`?GL_LUMINANCE_ALPHA` A R
`?GL_INTENSITY` R
`?GL_RGB` R G B
`?GL_RGBA` R G B A

Finally, the red, green, blue, alpha, luminance, and/or intensity components of the resulting pixels are stored in the color table. They form a one-dimensional table with indices in the range [0 width-1].

See **external** documentation.

getColorTable(Target, Format, Type, Table) -> ok

Types:

Target = enum()
Format = enum()
Type = enum()
Table = mem()

Retrieve contents of a color lookup table

`gl:getColorTable` returns in `Table` the contents of the color table specified by `Target` . No pixel transfer operations are performed, but pixel storage modes that are applicable to *gl:readPixels/7* are performed.

If a non-zero named buffer object is bound to the `?GL_PIXEL_PACK_BUFFER` target (see *gl:bindBuffer/2*) while a histogram table is requested, `Table` is treated as a byte offset into the buffer object's data store.

Color components that are requested in the specified `Format` , but which are not included in the internal format of the color lookup table, are returned as zero. The assignments of internal color components to the components requested by `Format` are

`Internal ComponentResulting Component`
Red Red
Green Green
Blue Blue
Alpha Alpha

Luminance Red
Intensity Red

See **external** documentation.

getColorTableParameterfv(Target, Pname) -> {float(), float(), float(), float()}

Types:

Target = enum()

Pname = enum()

Get color lookup table parameters

Returns parameters specific to color table `Target` .

When `Pname` is set to `?GL_COLOR_TABLE_SCALE` or `?GL_COLOR_TABLE_BIAS`, `gl:getColorTableParameter` returns the color table scale or bias parameters for the table specified by `Target` . For these queries, `Target` must be set to `?GL_COLOR_TABLE` , `?GL_POST_CONVOLUTION_COLOR_TABLE`, or `?GL_POST_COLOR_MATRIX_COLOR_TABLE` and `Params` points to an array of four elements, which receive the scale or bias factors for red, green, blue, and alpha, in that order.

`gl:getColorTableParameter` can also be used to retrieve the format and size parameters for a color table. For these queries, set `Target` to either the color table target or the proxy color table target. The format and size parameters are set by *gl:colorTable/6* .

The following table lists the format and size parameters that may be queried. For each symbolic constant listed below for `Pname` , `Params` must point to an array of the given length and receive the values indicated.

`ParameterNMeaning`
`?GL_COLOR_TABLE_FORMAT` 1 Internal format (e.g., `?GL_RGBA`)
`?GL_COLOR_TABLE_WIDTH` 1 Number of elements in table
`?GL_COLOR_TABLE_RED_SIZE` 1 Size of red component, in bits
`?GL_COLOR_TABLE_GREEN_SIZE` 1 Size of green component
`?GL_COLOR_TABLE_BLUE_SIZE` 1 Size of blue component
`?GL_COLOR_TABLE_ALPHA_SIZE` 1 Size of alpha component
`?GL_COLOR_TABLE_LUMINANCE_SIZE` 1 Size of luminance component
`?GL_COLOR_TABLE_INTENSITY_SIZE` 1 Size of intensity component

See **external** documentation.

getColorTableParameteriv(Target, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Target = enum()

Pname = enum()

See *getColorTableParameterfv/2*

colorSubTable(Target, Start, Count, Format, Type, Data) -> ok

Types:

Target = enum()

Start = integer()

Count = integer()

Format = enum()

```
Type = enum()
Data = offset() | mem()
```

Respecify a portion of a color table

`gl:colorSubTable` is used to respecify a contiguous portion of a color table previously defined using *gl:colorTable/6* . The pixels referenced by `Data` replace the portion of the existing table from indices `Start` to start +count-1, inclusive. This region may not include any entries outside the range of the color table as it was originally specified. It is not an error to specify a subtexture with width of 0, but such a specification has no effect.

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a portion of a color table is respecified, `Data` is treated as a byte offset into the buffer object's data store.

See **external** documentation.

copyColorSubTable(Target, Start, X, Y, Width) -> ok

Types:

```
Target = enum()
Start = integer()
X = integer()
Y = integer()
Width = integer()
```

Respecify a portion of a color table

`gl:copyColorSubTable` is used to respecify a contiguous portion of a color table previously defined using *gl:colorTable/6* . The pixels copied from the framebuffer replace the portion of the existing table from indices `Start` to start+x-1, inclusive. This region may not include any entries outside the range of the color table, as was originally specified. It is not an error to specify a subtexture with width of 0, but such a specification has no effect.

See **external** documentation.

convolutionFilter1D(Target, Internalformat, Width, Format, Type, Image) -> ok

Types:

```
Target = enum()
Internalformat = enum()
Width = integer()
Format = enum()
Type = enum()
Image = offset() | mem()
```

Define a one-dimensional convolution filter

`gl:convolutionFilter1D` builds a one-dimensional convolution filter kernel from an array of pixels.

The pixel array specified by `Width` , `Format` , `Type` , and `Data` is extracted from memory and processed just as if *gl:drawPixels/5* were called, but processing stops after the final expansion to RGBA is completed.

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a convolution filter is specified, `Data` is treated as a byte offset into the buffer object's data store.

The R, G, B, and A components of each pixel are next scaled by the four 1D `?GL_CONVOLUTION_FILTER_SCALE` parameters and biased by the four 1D `?GL_CONVOLUTION_FILTER_BIAS` parameters. (The scale and bias parameters are set by *gl:convolutionParameterf/3* using the `?GL_CONVOLUTION_1D` target and the names `?GL_CONVOLUTION_FILTER_SCALE` and `?GL_CONVOLUTION_FILTER_BIAS` . The parameters themselves

are vectors of four values that are applied to red, green, blue, and alpha, in that order.) The R, G, B, and A values are not clamped to [0,1] at any time during this process.

Each pixel is then converted to the internal format specified by `Internalformat` . This conversion simply maps the component values of the pixel (R, G, B, and A) to the values included in the internal format (red, green, blue, alpha, luminance, and intensity). The mapping is as follows:

`Internal FormatRedGreen BlueAlphaLuminanceIntensity`
`?GL_ALPHA` A
`?GL_LUMINANCE` R
`?GL_LUMINANCE_ALPHA` A R
`?GL_INTENSITY` R
`?GL_RGB` R G B
`?GL_RGBA` R G B A

The red, green, blue, alpha, luminance, and/or intensity components of the resulting pixels are stored in floating-point rather than integer format. They form a one-dimensional filter kernel image indexed with coordinate `i` such that `i` starts at 0 and increases from left to right. Kernel location `i` is derived from the `i`th pixel, counting from 0.

Note that after a convolution is performed, the resulting color components are also scaled by their corresponding `?GL_POST_CONVOLUTION_c_SCALE` parameters and biased by their corresponding `?GL_POST_CONVOLUTION_c_BIAS` parameters (where `c` takes on the values `RED`, `GREEN`, `BLUE`, and `ALPHA`). These parameters are set by *gl:pixelTransferf/2* .

See **external** documentation.

convolutionFilter2D(Target, Internalformat, Width, Height, Format, Type, Image) -> ok

Types:

```
Target = enum()
Internalformat = enum()
Width = integer()
Height = integer()
Format = enum()
Type = enum()
Image = offset() | mem()
```

Define a two-dimensional convolution filter

`gl:convolutionFilter2D` builds a two-dimensional convolution filter kernel from an array of pixels.

The pixel array specified by `Width` , `Height` , `Format` , `Type` , and `Data` is extracted from memory and processed just as if *gl:drawPixels/5* were called, but processing stops after the final expansion to RGBA is completed.

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a convolution filter is specified, `Data` is treated as a byte offset into the buffer object's data store.

The R, G, B, and A components of each pixel are next scaled by the four 2D `?GL_CONVOLUTION_FILTER_SCALE` parameters and biased by the four 2D `?GL_CONVOLUTION_FILTER_BIAS` parameters. (The scale and bias parameters are set by *gl:convolutionParameterf/3* using the `?GL_CONVOLUTION_2D` target and the names `?GL_CONVOLUTION_FILTER_SCALE` and `?GL_CONVOLUTION_FILTER_BIAS` . The parameters themselves are vectors of four values that are applied to red, green, blue, and alpha, in that order.) The R, G, B, and A values are not clamped to [0,1] at any time during this process.

Each pixel is then converted to the internal format specified by Internalformat . This conversion simply maps the component values of the pixel (R, G, B, and A) to the values included in the internal format (red, green, blue, alpha, luminance, and intensity). The mapping is as follows:

Internal FormatRedGreen BlueAlphaLuminanceIntensity
?GL_ALPHA A
?GL_LUMINANCE R
?GL_LUMINANCE_ALPHA A R
?GL_INTENSITY R
?GL_RGB R G B
?GL_RGBA R G B A

The red, green, blue, alpha, luminance, and/or intensity components of the resulting pixels are stored in floating-point rather than integer format. They form a two-dimensional filter kernel image indexed with coordinates i and j such that i starts at zero and increases from left to right, and j starts at zero and increases from bottom to top. Kernel location i,j is derived from the Nth pixel, where N is i+j* Width .

Note that after a convolution is performed, the resulting color components are also scaled by their corresponding ?GL_POST_CONVOLUTION_c_SCALE parameters and biased by their corresponding ? GL_POST_CONVOLUTION_c_BIAS parameters (where c takes on the values RED, GREEN, BLUE, and ALPHA). These parameters are set by *gl:pixelTransferf/2* .

See **external** documentation.

convolutionParameterf(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {float()}

Set convolution parameters

gl:convolutionParameter sets the value of a convolution parameter.

Target selects the convolution filter to be affected: ?GL_CONVOLUTION_1D, ?GL_CONVOLUTION_2D , or ? GL_SEPARABLE_2D for the 1D, 2D, or separable 2D filter, respectively.

Pname selects the parameter to be changed. ?GL_CONVOLUTION_FILTER_SCALE and ? GL_CONVOLUTION_FILTER_BIAS affect the definition of the convolution filter kernel; see *gl:convolutionFilter1D/6* , *gl:convolutionFilter2D/7* , and *gl:separableFilter2D/8* for details. In these cases, Params v is an array of four values to be applied to red, green, blue, and alpha values, respectively. The initial value for ?GL_CONVOLUTION_FILTER_SCALE is (1, 1, 1, 1), and the initial value for ? GL_CONVOLUTION_FILTER_BIAS is (0, 0, 0, 0).

A Pname value of ?GL_CONVOLUTION_BORDER_MODE controls the convolution border mode. The accepted modes are:

?GL_REDUCE: The image resulting from convolution is smaller than the source image. If the filter width is Wf and height is Hf, and the source image width is Ws and height is Hs, then the convolved image width will be Ws-Wf+1 and height will be Hs-Hf +1. (If this reduction would generate an image with zero or negative width and/or height, the output is simply null, with no error generated.) The coordinates of the image resulting from convolution are zero through Ws-Wf in width and zero through Hs-Hf in height.

?GL_CONSTANT_BORDER: The image resulting from convolution is the same size as the source image, and processed as if the source image were surrounded by pixels with their color specified by the ? GL_CONVOLUTION_BORDER_COLOR.

?GL_REPLICATE_BORDER: The image resulting from convolution is the same size as the source image, and processed as if the outermost pixel on the border of the source image were replicated.

See **external** documentation.

convolutionParameterfv(Target::enum(), Pname::enum(), Params) -> ok

Types:

Params = {Params::{float()}}

Equivalent to *convolutionParameterf(Target, Pname, Params)*.

convolutionParameteri(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {integer()}

See *convolutionParameterf/3*

convolutionParameteriv(Target::enum(), Pname::enum(), Params) -> ok

Types:

Params = {Params::{integer()}}

Equivalent to *convolutionParameteri(Target, Pname, Params)*.

copyConvolutionFilter1D(Target, Internalformat, X, Y, Width) -> ok

Types:

Target = enum()
Internalformat = enum()
X = integer()
Y = integer()
Width = integer()

Copy pixels into a one-dimensional convolution filter

gl:copyConvolutionFilter1D defines a one-dimensional convolution filter kernel with pixels from the current ?GL_READ_BUFFER (rather than from main memory, as is the case for *gl:convolutionFilter1D/6*).

The screen-aligned pixel rectangle with lower-left corner at (X , Y), width Width and height 1 is used to define the convolution filter. If any pixels within this region are outside the window that is associated with the GL context, the values obtained for those pixels are undefined.

The pixels in the rectangle are processed exactly as if *gl:readPixels/7* had been called with format set to RGBA, but the process stops just before final conversion. The R, G, B, and A components of each pixel are next scaled by the four 1D ?GL_CONVOLUTION_FILTER_SCALE parameters and biased by the four 1D ?GL_CONVOLUTION_FILTER_BIAS parameters. (The scale and bias parameters are set by *gl:convolutionParameterf/3* using the ?GL_CONVOLUTION_1D target and the names ?GL_CONVOLUTION_FILTER_SCALE and ?GL_CONVOLUTION_FILTER_BIAS . The parameters themselves are vectors of four values that are applied to red, green, blue, and alpha, in that order.) The R, G, B, and A values are not clamped to [0,1] at any time during this process.

Each pixel is then converted to the internal format specified by `Internalformat` . This conversion simply maps the component values of the pixel (R, G, B, and A) to the values included in the internal format (red, green, blue, alpha, luminance, and intensity). The mapping is as follows:

`Internal FormatRedGreen BlueAlphaLuminanceIntensity`
`?GL_ALPHA` A
`?GL_LUMINANCE` R
`?GL_LUMINANCE_ALPHA` A R
`?GL_INTENSITY` R
`?GL_RGB` R G B
`?GL_RGBA` R G B A

The red, green, blue, alpha, luminance, and/or intensity components of the resulting pixels are stored in floating-point rather than integer format.

Pixel ordering is such that lower x screen coordinates correspond to lower `i` filter image coordinates.

Note that after a convolution is performed, the resulting color components are also scaled by their corresponding `?GL_POST_CONVOLUTION_c_SCALE` parameters and biased by their corresponding `?GL_POST_CONVOLUTION_c_BIAS` parameters (where `c` takes on the values `RED`, `GREEN`, `BLUE`, and `ALPHA`). These parameters are set by *gl:pixelTransferf/2* .

See **external** documentation.

`copyConvolutionFilter2D(Target, Internalformat, X, Y, Width, Height) -> ok`

Types:

```
Target = enum()
Internalformat = enum()
X = integer()
Y = integer()
Width = integer()
Height = integer()
```

Copy pixels into a two-dimensional convolution filter

`gl:copyConvolutionFilter2D` defines a two-dimensional convolution filter kernel with pixels from the current `?GL_READ_BUFFER` (rather than from main memory, as is the case for *gl:convolutionFilter2D/7*).

The screen-aligned pixel rectangle with lower-left corner at (`X` , `Y`), width `Width` and height `Height` is used to define the convolution filter. If any pixels within this region are outside the window that is associated with the GL context, the values obtained for those pixels are undefined.

The pixels in the rectangle are processed exactly as if *gl:readPixels/7* had been called with `format` set to RGBA, but the process stops just before final conversion. The R, G, B, and A components of each pixel are next scaled by the four 2D `?GL_CONVOLUTION_FILTER_SCALE` parameters and biased by the four 2D `?GL_CONVOLUTION_FILTER_BIAS` parameters. (The scale and bias parameters are set by *gl:convolutionParameterf/3* using the `?GL_CONVOLUTION_2D` target and the names `?GL_CONVOLUTION_FILTER_SCALE` and `?GL_CONVOLUTION_FILTER_BIAS` . The parameters themselves are vectors of four values that are applied to red, green, blue, and alpha, in that order.) The R, G, B, and A values are not clamped to [0,1] at any time during this process.

Each pixel is then converted to the internal format specified by `Internalformat` . This conversion simply maps the component values of the pixel (R, G, B, and A) to the values included in the internal format (red, green, blue, alpha, luminance, and intensity). The mapping is as follows:

`Internal FormatRedGreen BlueAlphaLuminanceIntensity`
`?GL_ALPHA` A

?GL_LUMINANCE R
?GL_LUMINANCE_ALPHA A R
?GL_INTENSITY R
?GL_RGB R G B
?GL_RGBA R G B A

The red, green, blue, alpha, luminance, and/or intensity components of the resulting pixels are stored in floating-point rather than integer format.

Pixel ordering is such that lower x screen coordinates correspond to lower i filter image coordinates, and lower y screen coordinates correspond to lower j filter image coordinates.

Note that after a convolution is performed, the resulting color components are also scaled by their corresponding ?GL_POST_CONVOLUTION_c_SCALE parameters and biased by their corresponding ?GL_POST_CONVOLUTION_c_BIAS parameters (where c takes on the values RED, GREEN, BLUE, and ALPHA). These parameters are set by *gl:pixelTransferf/2* .

See **external** documentation.

getConvolutionFilter(Target, Format, Type, Image) -> ok

Types:

Target = enum()
Format = enum()
Type = enum()
Image = mem()

Get current 1D or 2D convolution filter kernel

gl:getConvolutionFilter returns the current 1D or 2D convolution filter kernel as an image. The one- or two-dimensional image is placed in Image according to the specifications in Format and Type . No pixel transfer operations are performed on this image, but the relevant pixel storage modes are applied.

If a non-zero named buffer object is bound to the ?GL_PIXEL_PACK_BUFFER target (see *gl:bindBuffer/2*) while a convolution filter is requested, Image is treated as a byte offset into the buffer object's data store.

Color components that are present in Format but not included in the internal format of the filter are returned as zero. The assignments of internal color components to the components of Format are as follows.

Internal Component Resulting Component
Red Red
Green Green
Blue Blue
Alpha Alpha
Luminance Red
Intensity Red

See **external** documentation.

getConvolutionParameterfv(Target, Pname) -> {float(), float(), float(), float()}

Types:

Target = enum()
Pname = enum()

Get convolution parameters

gl:getConvolutionParameter retrieves convolution parameters. Target determines which convolution filter is queried. Pname determines which parameter is returned:

?GL_CONVOLUTION_BORDER_MODE: The convolution border mode. See *gl:convolutionParameterf/3* for a list of border modes.

?GL_CONVOLUTION_BORDER_COLOR: The current convolution border color. Params must be a pointer to an array of four elements, which will receive the red, green, blue, and alpha border colors.

?GL_CONVOLUTION_FILTER_SCALE: The current filter scale factors. Params must be a pointer to an array of four elements, which will receive the red, green, blue, and alpha filter scale factors in that order.

?GL_CONVOLUTION_FILTER_BIAS: The current filter bias factors. Params must be a pointer to an array of four elements, which will receive the red, green, blue, and alpha filter bias terms in that order.

?GL_CONVOLUTION_FORMAT: The current internal format. See *gl:convolutionFilter1D/6* , *gl:convolutionFilter2D/7* , and *gl:separableFilter2D/8* for lists of allowable formats.

?GL_CONVOLUTION_WIDTH: The current filter image width.

?GL_CONVOLUTION_HEIGHT: The current filter image height.

?GL_MAX_CONVOLUTION_WIDTH: The maximum acceptable filter image width.

?GL_MAX_CONVOLUTION_HEIGHT: The maximum acceptable filter image height.

See **external** documentation.

getConvolutionParameteriv(Target, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Target = enum()

Pname = enum()

See *getConvolutionParameterfv/2*

separableFilter2D(Target, Internalformat, Width, Height, Format, Type, Row, Column) -> ok

Types:

Target = enum()

Internalformat = enum()

Width = integer()

Height = integer()

Format = enum()

Type = enum()

Row = offset() | mem()

Column = offset() | mem()

Define a separable two-dimensional convolution filter

gl:separableFilter2D builds a two-dimensional separable convolution filter kernel from two arrays of pixels.

The pixel arrays specified by (Width , Format , Type , Row) and (Height , Format , Type , Column) are processed just as if they had been passed to *gl:drawPixels/5* , but processing stops after the final expansion to RGBA is completed.

If a non-zero named buffer object is bound to the ?GL_PIXEL_UNPACK_BUFFER target (see *gl:bindBuffer/2*) while a convolution filter is specified, Row and Column are treated as byte offsets into the buffer object's data store.

Next, the R, G, B, and A components of all pixels in both arrays are scaled by the four separable 2D ?GL_CONVOLUTION_FILTER_SCALE parameters and biased by the four separable 2D ?GL_CONVOLUTION_FILTER_BIAS parameters. (The scale and bias parameters are set by *gl:convolutionParameterf/3* using the ?GL_SEPARABLE_2D target and the names ?GL_CONVOLUTION_FILTER_SCALE and ?GL_CONVOLUTION_FILTER_BIAS. The parameters themselves are vectors of four values that are applied to red, green, blue, and alpha, in that order.) The R, G, B, and A values are not clamped to [0,1] at any time during this process.

Each pixel is then converted to the internal format specified by Internalformat . This conversion simply maps the component values of the pixel (R, G, B, and A) to the values included in the internal format (red, green, blue, alpha, luminance, and intensity). The mapping is as follows:

Internal FormatRed GreenBlueAlphaLuminance Intensity
?GL_LUMINANCE R
?GL_LUMINANCE_ALPHA A R
?GL_INTENSITY R
?GL_RGB R G B
?GL_RGBA R G B A

The red, green, blue, alpha, luminance, and/or intensity components of the resulting pixels are stored in floating-point rather than integer format. They form two one-dimensional filter kernel images. The row image is indexed by coordinate i starting at zero and increasing from left to right. Each location in the row image is derived from element i of Row . The column image is indexed by coordinate j starting at zero and increasing from bottom to top. Each location in the column image is derived from element j of Column .

Note that after a convolution is performed, the resulting color components are also scaled by their corresponding ?GL_POST_CONVOLUTION_c_SCALE parameters and biased by their corresponding ?GL_POST_CONVOLUTION_c_BIAS parameters (where c takes on the values RED, GREEN, BLUE, and ALPHA). These parameters are set by *gl:pixelTransferf/2* .

See **external** documentation.

getHistogram(Target, Reset, Format, Type, Values) -> ok

Types:

```
Target = enum()
Reset = 0 | 1
Format = enum()
Type = enum()
Values = mem()
```

Get histogram table

gl:getHistogram returns the current histogram table as a one-dimensional image with the same width as the histogram. No pixel transfer operations are performed on this image, but pixel storage modes that are applicable to 1D images are honored.

If a non-zero named buffer object is bound to the ?GL_PIXEL_PACK_BUFFER target (see *gl:bindBuffer/2*) while a histogram table is requested, Values is treated as a byte offset into the buffer object's data store.

Color components that are requested in the specified Format , but which are not included in the internal format of the histogram, are returned as zero. The assignments of internal color components to the components requested by Format are:

Internal ComponentResulting Component
Red Red
Green Green

Blue Blue
Alpha Alpha
Luminance Red

See **external** documentation.

getHistogramParameterfv(Target, Pname) -> {float()}

Types:

```
Target = enum()
Pname = enum()
```

Get histogram parameters

`gl:getHistogramParameter` is used to query parameter values for the current histogram or for a proxy. The histogram state information may be queried by calling `gl:getHistogramParameter` with a `Target` of `?GL_HISTOGRAM` (to obtain information for the current histogram table) or `?GL_PROXY_HISTOGRAM` (to obtain information from the most recent proxy request) and one of the following values for the `Pname` argument:

`ParameterDescription`
`?GL_HISTOGRAM_WIDTH` Histogram table width
`?GL_HISTOGRAM_FORMAT` Internal format
`?GL_HISTOGRAM_RED_SIZE` Red component counter size, in bits
`?GL_HISTOGRAM_GREEN_SIZE` Green component counter size, in bits
`?GL_HISTOGRAM_BLUE_SIZE` Blue component counter size, in bits
`?GL_HISTOGRAM_ALPHA_SIZE` Alpha component counter size, in bits
`?GL_HISTOGRAM_LUMINANCE_SIZE` Luminance component counter size, in bits
`?GL_HISTOGRAM_SINK` Value of the `sink` parameter

See **external** documentation.

getHistogramParameteriv(Target, Pname) -> {integer()}

Types:

```
Target = enum()
Pname = enum()
```

See *getHistogramParameterfv/2*

getMinmax(Target, Reset, Format, Types, Values) -> ok

Types:

```
Target = enum()
Reset = 0 | 1
Format = enum()
Types = enum()
Values = mem()
```

Get minimum and maximum pixel values

`gl:getMinmax` returns the accumulated minimum and maximum pixel values (computed on a per-component basis) in a one-dimensional image of width 2. The first set of return values are the minima, and the second set of return values are the maxima. The format of the return values is determined by `Format` , and their type is determined by `Types` .

If a non-zero named buffer object is bound to the `?GL_PIXEL_PACK_BUFFER` target (see *gl:bindBuffer/2*) while minimum and maximum pixel values are requested, `Values` is treated as a byte offset into the buffer object's data store.

No pixel transfer operations are performed on the return values, but pixel storage modes that are applicable to one-dimensional images are performed. Color components that are requested in the specified `Format` , but that are not included in the internal format of the minmax table, are returned as zero. The assignment of internal color components to the components requested by `Format` are as follows:

`Internal ComponentResulting Component`
Red Red
Green Green
Blue Blue
Alpha Alpha
Luminance Red

If `Reset` is `?GL_TRUE`, the minmax table entries corresponding to the return values are reset to their initial values. Minimum and maximum values that are not returned are not modified, even if `Reset` is `?GL_TRUE`.

See **external** documentation.

`getMinmaxParameterfv(Target, Pname) -> {float()}`

Types:

`Target = enum()`
`Pname = enum()`

Get minmax parameters

`gl:getMinmaxParameter` retrieves parameters for the current minmax table by setting `Pname` to one of the following values:

`ParameterDescription`
`?GL_MINMAX_FORMAT` Internal format of minmax table
`?GL_MINMAX_SINK` Value of the `sink` parameter

See **external** documentation.

`getMinmaxParameteriv(Target, Pname) -> {integer()}`

Types:

`Target = enum()`
`Pname = enum()`

See *getMinmaxParameterfv/2*

`histogram(Target, Width, Internalformat, Sink) -> ok`

Types:

`Target = enum()`
`Width = integer()`
`Internalformat = enum()`
`Sink = 0 | 1`

Define histogram table

When `?GL_HISTOGRAM` is enabled, RGBA color components are converted to histogram table indices by clamping to the range [0,1], multiplying by the width of the histogram table, and rounding to the nearest integer. The table entries selected by the RGBA indices are then incremented. (If the internal format of the histogram table includes luminance, then the index derived from the R color component determines the luminance table entry to be incremented.) If a histogram table entry is incremented beyond its maximum value, then its value becomes undefined. (This is not an error.)

Histogramming is performed only for RGBA pixels (though these may be specified originally as color indices and converted to RGBA by index table lookup). Histogramming is enabled with *gl:enable/1* and disabled with *gl:enable/1* .

When `Target` is `?GL_HISTOGRAM`, `gl:histogram` redefines the current histogram table to have `Width` entries of the format specified by `Internalformat` . The entries are indexed 0 through width-1, and all entries are initialized to zero. The values in the previous histogram table, if any, are lost. If `Sink` is `?GL_TRUE` , then pixels are discarded after histogramming; no further processing of the pixels takes place, and no drawing, texture loading, or pixel readback will result.

When `Target` is `?GL_PROXY_HISTOGRAM`, `gl:histogram` computes all state information as if the histogram table were to be redefined, but does not actually define the new table. If the requested histogram table is too large to be supported, then the state information will be set to zero. This provides a way to determine if a histogram table with the given parameters can be supported.

See **external** documentation.

`minmax(Target, Internalformat, Sink) -> ok`

Types:

```
Target = enum()
Internalformat = enum()
Sink = 0 | 1
```

Define minmax table

When `?GL_MINMAX` is enabled, the RGBA components of incoming pixels are compared to the minimum and maximum values for each component, which are stored in the two-element minmax table. (The first element stores the minima, and the second element stores the maxima.) If a pixel component is greater than the corresponding component in the maximum element, then the maximum element is updated with the pixel component value. If a pixel component is less than the corresponding component in the minimum element, then the minimum element is updated with the pixel component value. (In both cases, if the internal format of the minmax table includes luminance, then the R color component of incoming pixels is used for comparison.) The contents of the minmax table may be retrieved at a later time by calling *gl:getMinmax/5* . The minmax operation is enabled or disabled by calling *gl:enable/1* or *gl:enable/1* , respectively, with an argument of `?GL_MINMAX` .

`gl:minmax` redefines the current minmax table to have entries of the format specified by `Internalformat` . The maximum element is initialized with the smallest possible component values, and the minimum element is initialized with the largest possible component values. The values in the previous minmax table, if any, are lost. If `Sink` is `?GL_TRUE` , then pixels are discarded after minmax; no further processing of the pixels takes place, and no drawing, texture loading, or pixel readback will result.

See **external** documentation.

`resetHistogram(Target) -> ok`

Types:

```
Target = enum()
```

Reset histogram table entries to zero

`gl:resetHistogram` resets all the elements of the current histogram table to zero.

See **external** documentation.

`resetMinmax(Target) -> ok`

Types:

```
Target = enum()
```

Reset minmax table entries to initial values

`gl:resetMinmax` resets the elements of the current minmax table to their initial values: the `maximum` element receives the minimum possible component values, and the `minimum` element receives the maximum possible component values.

See **external** documentation.

activeTexture(Texture) -> ok

Types:

```
Texture = enum()
```

Select active texture unit

`gl:activeTexture` selects which texture unit subsequent texture state calls will affect. The number of texture units an implementation supports is implementation dependent, but must be at least 80.

See **external** documentation.

sampleCoverage(Value, Invert) -> ok

Types:

```
Value = clamp()
Invert = 0 | 1
```

Specify multisample coverage parameters

Multisampling samples a pixel multiple times at various implementation-dependent subpixel locations to generate antialiasing effects. Multisampling transparently antialiases points, lines, polygons, and images if it is enabled.

`Value` is used in constructing a temporary mask used in determining which samples will be used in resolving the final fragment color. This mask is bitwise-anded with the coverage mask generated from the multisampling computation. If the `Invert` flag is set, the temporary mask is inverted (all bits flipped) and then the bitwise-and is computed.

If an implementation does not have any multisample buffers available, or multisampling is disabled, rasterization occurs with only a single sample computing a pixel's final RGB color.

Provided an implementation supports multisample buffers, and multisampling is enabled, then a pixel's final color is generated by combining several samples per pixel. Each sample contains color, depth, and stencil information, allowing those operations to be performed on each sample.

See **external** documentation.

compressedTexImage3D(Target, Level, Internalformat, Width, Height, Depth, Border, ImageSize, Data) -> ok

Types:

```
Target = enum()
Level = integer()
Internalformat = enum()
Width = integer()
Height = integer()
Depth = integer()
Border = integer()
ImageSize = integer()
Data = offset() | mem()
```

Specify a three-dimensional texture image in a compressed format

Texturing allows elements of an image array to be read by shaders.

`gl:compressedTexImage3D` loads a previously defined, and retrieved, compressed three-dimensional texture image if `Target` is `?GL_TEXTURE_3D` (see *gl:texImage3D/10*).

If `Target` is `?GL_TEXTURE_2D_ARRAY`, `Data` is treated as an array of compressed 2D textures.

If `Target` is `?GL_PROXY_TEXTURE_3D` or `?GL_PROXY_TEXTURE_2D_ARRAY`, no data is read from `Data` , but all of the texture image state is recalculated, checked for consistency, and checked against the implementation's capabilities. If the implementation cannot handle a texture of the requested texture size, it sets all of the image state to 0, but does not generate an error (see *gl:getError/0*). To query for an entire mipmap array, use an image array level greater than or equal to 1.

`Internalformat` must be a known compressed image format (such as `?GL_RGTC`) or an extension-specified compressed-texture format. When a texture is loaded with *gl:texImage2D/9* using a generic compressed texture format (e.g., `?GL_COMPRESSED_RGB`), the GL selects from one of its extensions supporting compressed textures. In order to load the compressed texture image using `gl:compressedTexImage3D`, query the compressed texture image's size and format using *gl:getTexLevelParameterfv/3* .

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a texture image is specified, `Data` is treated as a byte offset into the buffer object's data store.

If the compressed data are arranged into fixed-size blocks of texels, the pixel storage modes can be used to select a sub-rectangle from a larger containing rectangle. These pixel storage modes operate in the same way as they do for *gl:texImage1D/8* . In the following description, denote by b s, b w, b h, and b d, the values of pixel storage modes `?GL_UNPACK_COMPRESSED_BLOCK_SIZE`, `?GL_UNPACK_COMPRESSED_BLOCK_WIDTH`, `?GL_UNPACK_COMPRESSED_BLOCK_HEIGHT` , and `?GL_UNPACK_COMPRESSED_BLOCK_DEPTH`, respectively. b s is the compressed block size in bytes; b w, b h, and b d are the compressed block width, height, and depth in pixels.

By default the pixel storage modes `?GL_UNPACK_ROW_LENGTH`, `?GL_UNPACK_SKIP_ROWS` , `?GL_UNPACK_SKIP_PIXELS`, `?GL_UNPACK_IMAGE_HEIGHT` and `?GL_UNPACK_SKIP_IMAGES` are ignored for compressed images. To enable `?GL_UNPACK_SKIP_PIXELS` and `?GL_UNPACK_ROW_LENGTH` , b s and b w must both be non-zero. To also enable `?GL_UNPACK_SKIP_ROWS` and `?GL_UNPACK_IMAGE_HEIGHT` , b h must be non-zero. To also enable `?GL_UNPACK_SKIP_IMAGES`, b d must be non-zero. All parameters must be consistent with the compressed format to produce the desired results.

When selecting a sub-rectangle from a compressed image: the value of `?GL_UNPACK_SKIP_PIXELS` must be a multiple of b w;the value of `?GL_UNPACK_SKIP_ROWS` must be a multiple of b w;the value of `?GL_UNPACK_SKIP_IMAGES` must be a multiple of b w.

`ImageSize` must be equal to:

b s*|width b/w|*|height b/h|*|depth b/d|

See **external** documentation.

`compressedTexImage2D(Target, Level, Internalformat, Width, Height, Border, ImageSize, Data) -> ok`

Types:

```
Target = enum()
Level = integer()
Internalformat = enum()
Width = integer()
Height = integer()
```

```
Border = integer()
ImageSize = integer()
Data = offset() | mem()
```

Specify a two-dimensional texture image in a compressed format

Texturing allows elements of an image array to be read by shaders.

`gl:compressedTexImage2D` loads a previously defined, and retrieved, compressed two-dimensional texture image if `Target` is `?GL_TEXTURE_2D`, or one of the cube map faces such as `?GL_TEXTURE_CUBE_MAP_POSITIVE_X`. (see *gl:texImage2D/9*).

If `Target` is `?GL_TEXTURE_1D_ARRAY`, `Data` is treated as an array of compressed 1D textures.

If `Target` is `?GL_PROXY_TEXTURE_2D`, `?GL_PROXY_TEXTURE_1D_ARRAY` or `?GL_PROXY_CUBE_MAP` , no data is read from `Data` , but all of the texture image state is recalculated, checked for consistency, and checked against the implementation's capabilities. If the implementation cannot handle a texture of the requested texture size, it sets all of the image state to 0, but does not generate an error (see *gl:getError/0*). To query for an entire mipmap array, use an image array level greater than or equal to 1.

`Internalformat` must be a known compressed image format (such as `?GL_RGTC`) or an extension-specified compressed-texture format. When a texture is loaded with *gl:texImage2D/9* using a generic compressed texture format (e.g., `?GL_COMPRESSED_RGB`), the GL selects from one of its extensions supporting compressed textures. In order to load the compressed texture image using `gl:compressedTexImage2D`, query the compressed texture image's size and format using *gl:getTexLevelParameterfv/3* .

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a texture image is specified, `Data` is treated as a byte offset into the buffer object's data store.

If the compressed data are arranged into fixed-size blocks of texels, the pixel storage modes can be used to select a sub-rectangle from a larger containing rectangle. These pixel storage modes operate in the same way as they do for *gl:texImage2D/9* . In the following description, denote by b s, b w, b h, and b d, the values of pixel storage modes `?GL_UNPACK_COMPRESSED_BLOCK_SIZE`, `?GL_UNPACK_COMPRESSED_BLOCK_WIDTH`, `?GL_UNPACK_COMPRESSED_BLOCK_HEIGHT` , and `?GL_UNPACK_COMPRESSED_BLOCK_DEPTH`, respectively. b s is the compressed block size in bytes; b w, b h, and b d are the compressed block width, height, and depth in pixels.

By default the pixel storage modes `?GL_UNPACK_ROW_LENGTH`, `?GL_UNPACK_SKIP_ROWS` , `?GL_UNPACK_SKIP_PIXELS`, `?GL_UNPACK_IMAGE_HEIGHT` and `?GL_UNPACK_SKIP_IMAGES` are ignored for compressed images. To enable `?GL_UNPACK_SKIP_PIXELS` and `?GL_UNPACK_ROW_LENGTH` , b s and b w must both be non-zero. To also enable `?GL_UNPACK_SKIP_ROWS` and `?GL_UNPACK_IMAGE_HEIGHT` , b h must be non-zero. To also enable `?GL_UNPACK_SKIP_IMAGES`, b d must be non-zero. All parameters must be consistent with the compressed format to produce the desired results.

When selecting a sub-rectangle from a compressed image: the value of `?GL_UNPACK_SKIP_PIXELS` must be a multiple of b w;the value of `?GL_UNPACK_SKIP_ROWS` must be a multiple of b w.

`ImageSize` must be equal to:

b s*|width b/w|*|height b/h|

See **external** documentation.

`compressedTexImage1D(Target, Level, Internalformat, Width, Border, ImageSize, Data) -> ok`

Types:

```
Target = enum()
Level = integer()
```

```
Internalformat = enum()
Width = integer()
Border = integer()
ImageSize = integer()
Data = offset() | mem()
```

Specify a one-dimensional texture image in a compressed format

Texturing allows elements of an image array to be read by shaders.

`gl:compressedTexImage1D` loads a previously defined, and retrieved, compressed one-dimensional texture image if `Target` is `?GL_TEXTURE_1D` (see *gl:texImage1D/8*).

If `Target` is `?GL_PROXY_TEXTURE_1D`, no data is read from `Data` , but all of the texture image state is recalculated, checked for consistency, and checked against the implementation's capabilities. If the implementation cannot handle a texture of the requested texture size, it sets all of the image state to 0, but does not generate an error (see *gl:getError/0*). To query for an entire mipmap array, use an image array level greater than or equal to 1.

`Internalformat` must be an extension-specified compressed-texture format. When a texture is loaded with *gl:texImage1D/8* using a generic compressed texture format (e.g., `?GL_COMPRESSED_RGB`) the GL selects from one of its extensions supporting compressed textures. In order to load the compressed texture image using `gl:compressedTexImage1D` , query the compressed texture image's size and format using *gl:getTexLevelParameterfv/3* .

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a texture image is specified, `Data` is treated as a byte offset into the buffer object's data store.

If the compressed data are arranged into fixed-size blocks of texels, the pixel storage modes can be used to select a sub-rectangle from a larger containing rectangle. These pixel storage modes operate in the same way as they do for *gl:texImage1D/8* . In the following description, denote by b s, b w, b h, and b d, the values of pixel storage modes `?GL_UNPACK_COMPRESSED_BLOCK_SIZE`, `?GL_UNPACK_COMPRESSED_BLOCK_WIDTH`, `?GL_UNPACK_COMPRESSED_BLOCK_HEIGHT` , and `?GL_UNPACK_COMPRESSED_BLOCK_DEPTH`, respectively. b s is the compressed block size in bytes; b w, b h, and b d are the compressed block width, height, and depth in pixels.

By default the pixel storage modes `?GL_UNPACK_ROW_LENGTH`, `?GL_UNPACK_SKIP_ROWS` , `?GL_UNPACK_SKIP_PIXELS`, `?GL_UNPACK_IMAGE_HEIGHT` and `?GL_UNPACK_SKIP_IMAGES` are ignored for compressed images. To enable `?GL_UNPACK_SKIP_PIXELS` and `?GL_UNPACK_ROW_LENGTH` , b s and b w must both be non-zero. To also enable `?GL_UNPACK_SKIP_ROWS` and `?GL_UNPACK_IMAGE_HEIGHT` , b h must be non-zero. To also enable `?GL_UNPACK_SKIP_IMAGES`, b d must be non-zero. All parameters must be consistent with the compressed format to produce the desired results.

When selecting a sub-rectangle from a compressed image: the value of `?GL_UNPACK_SKIP_PIXELS` must be a multiple of b w;

`ImageSize` must be equal to:

b s*|width b/w|

See **external** documentation.

`compressedTexSubImage3D(Target, Level, Xoffset, Yoffset, Zoffset, Width, Height, Depth, Format, ImageSize, Data) -> ok`

Types:

```
Target = enum()
Level = integer()
Xoffset = integer()
```

```
Yoffset = integer()
Zoffset = integer()
Width = integer()
Height = integer()
Depth = integer()
Format = enum()
ImageSize = integer()
Data = offset() | mem()
```

Specify a three-dimensional texture subimage in a compressed format

Texturing allows elements of an image array to be read by shaders.

`gl:compressedTexSubImage3D` redefines a contiguous subregion of an existing three-dimensional texture image. The texels referenced by `Data` replace the portion of the existing texture array with x indices `Xoffset` and xoffset+width-1, and the y indices `Yoffset` and yoffset+height-1, and the z indices `Zoffset` and zoffset+depth-1, inclusive. This region may not include any texels outside the range of the texture array as it was originally specified. It is not an error to specify a subtexture with width of 0, but such a specification has no effect.

`Internalformat` must be a known compressed image format (such as `?GL_RGTC`) or an extension-specified compressed-texture format. The `Format` of the compressed texture image is selected by the GL implementation that compressed it (see *gl:texImage3D/10*) and should be queried at the time the texture was compressed with *gl:getTexLevelParameterfv/3* .

If a non-zero named buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target (see *gl:bindBuffer/2*) while a texture image is specified, `Data` is treated as a byte offset into the buffer object's data store.

See **external** documentation.

```
compressedTexSubImage2D(Target, Level, Xoffset, Yoffset, Width, Height,
Format, ImageSize, Data) -> ok
```

Types:

```
Target = enum()
Level = integer()
Xoffset = integer()
Yoffset = integer()
Width = integer()
Height = integer()
Format = enum()
ImageSize = integer()
Data = offset() | mem()
```

Specify a two-dimensional texture subimage in a compressed format

Texturing allows elements of an image array to be read by shaders.

`gl:compressedTexSubImage2D` redefines a contiguous subregion of an existing two-dimensional texture image. The texels referenced by `Data` replace the portion of the existing texture array with x indices `Xoffset` and xoffset+width-1, and the y indices `Yoffset` and yoffset+height-1, inclusive. This region may not include any texels outside the range of the texture array as it was originally specified. It is not an error to specify a subtexture with width of 0, but such a specification has no effect.

`Internalformat` must be a known compressed image format (such as `?GL_RGTC`) or an extension-specified compressed-texture format. The `Format` of the compressed texture image is selected by the GL implementation

that compressed it (see *gl:texImage2D/9*) and should be queried at the time the texture was compressed with *gl:getTexLevelParameterfv/3* .

If a non-zero named buffer object is bound to the ?GL_PIXEL_UNPACK_BUFFER target (see *gl:bindBuffer/2*) while a texture image is specified, Data is treated as a byte offset into the buffer object's data store.

See **external** documentation.

compressedTexSubImage1D(Target, Level, Xoffset, Width, Format, ImageSize, Data) -> ok

Types:

Target = enum()
Level = integer()
Xoffset = integer()
Width = integer()
Format = enum()
ImageSize = integer()
Data = offset() | mem()

Specify a one-dimensional texture subimage in a compressed format

Texturing allows elements of an image array to be read by shaders.

gl:compressedTexSubImage1D redefines a contiguous subregion of an existing one-dimensional texture image. The texels referenced by Data replace the portion of the existing texture array with x indices Xoffset and xoffset+width-1, inclusive. This region may not include any texels outside the range of the texture array as it was originally specified. It is not an error to specify a subtexture with width of 0, but such a specification has no effect.

Internalformat must be a known compressed image format (such as ?GL_RGTC) or an extension-specified compressed-texture format. The Format of the compressed texture image is selected by the GL implementation that compressed it (see *gl:texImage1D/8*), and should be queried at the time the texture was compressed with *gl:getTexLevelParameterfv/3* .

If a non-zero named buffer object is bound to the ?GL_PIXEL_UNPACK_BUFFER target (see *gl:bindBuffer/2*) while a texture image is specified, Data is treated as a byte offset into the buffer object's data store.

See **external** documentation.

getCompressedTexImage(Target, Lod, Img) -> ok

Types:

Target = enum()
Lod = integer()
Img = mem()

Return a compressed texture image

gl:getCompressedTexImage returns the compressed texture image associated with Target and Lod into Img . Img should be an array of ?GL_TEXTURE_COMPRESSED_IMAGE_SIZE bytes. Target specifies whether the desired texture image was one specified by *gl:texImage1D/8* (?GL_TEXTURE_1D), *gl:texImage2D/9* (?GL_TEXTURE_2D or any of ?GL_TEXTURE_CUBE_MAP_*), or *gl:texImage3D/10* (?GL_TEXTURE_3D). Lod specifies the level-of-detail number of the desired image.

If a non-zero named buffer object is bound to the ?GL_PIXEL_PACK_BUFFER target (see *gl:bindBuffer/2*) while a texture image is requested, Img is treated as a byte offset into the buffer object's data store.

To minimize errors, first verify that the texture is compressed by calling *gl:getTexLevelParameterfv/3* with argument `?GL_TEXTURE_COMPRESSED`. If the texture is compressed, then determine the amount of memory required to store the compressed texture by calling *gl:getTexLevelParameterfv/3* with argument `?GL_TEXTURE_COMPRESSED_IMAGE_SIZE`. Finally, retrieve the internal format of the texture by calling *gl:getTexLevelParameterfv/3* with argument `?GL_TEXTURE_INTERNAL_FORMAT` . To store the texture for later use, associate the internal format and size with the retrieved texture image. These data can be used by the respective texture or subtexture loading routine used for loading `Target` textures.

See **external** documentation.

clientActiveTexture(Texture) -> ok

Types:

Texture = enum()

Select active texture unit

`gl:clientActiveTexture` selects the vertex array client state parameters to be modified by *gl:texCoordPointer/4* , and enabled or disabled with *gl:enableClientState/1* or *gl:enableClientState/1* , respectively, when called with a parameter of `?GL_TEXTURE_COORD_ARRAY` .

See **external** documentation.

multiTexCoord1d(Target, S) -> ok

Types:

Target = enum()

S = float()

Set the current texture coordinates

`gl:multiTexCoord` specifies texture coordinates in one, two, three, or four dimensions. `gl:multiTexCoord1` sets the current texture coordinates to (s 0 0 1); a call to `gl:multiTexCoord2` sets them to (s t 0 1). Similarly, `gl:multiTexCoord3` specifies the texture coordinates as (s t r 1), and `gl:multiTexCoord4` defines all four components explicitly as (s t r q).

The current texture coordinates are part of the data that is associated with each vertex and with the current raster position. Initially, the values for (s t r q) are (0 0 0 1).

See **external** documentation.

multiTexCoord1dv(Target::enum(), V) -> ok

Types:

V = {S::float()}

Equivalent to *multiTexCoord1d(Target, S)*.

multiTexCoord1f(Target, S) -> ok

Types:

Target = enum()

S = float()

See *multiTexCoord1d/2*

multiTexCoord1fv(Target::enum(), V) -> ok

Types:

V = {S::float()}

Equivalent to *multiTexCoord1f(Target, S)*.

multiTexCoord1i(Target, S) -> ok

Types:

Target = enum()

S = integer()

See *multiTexCoord1d/2*

multiTexCoord1iv(Target::enum(), V) -> ok

Types:

V = {S::integer()}

Equivalent to *multiTexCoord1i(Target, S)*.

multiTexCoord1s(Target, S) -> ok

Types:

Target = enum()

S = integer()

See *multiTexCoord1d/2*

multiTexCoord1sv(Target::enum(), V) -> ok

Types:

V = {S::integer()}

Equivalent to *multiTexCoord1s(Target, S)*.

multiTexCoord2d(Target, S, T) -> ok

Types:

Target = enum()

S = float()

T = float()

See *multiTexCoord1d/2*

multiTexCoord2dv(Target::enum(), V) -> ok

Types:

V = {S::float(), T::float()}

Equivalent to *multiTexCoord2d(Target, S, T)*.

multiTexCoord2f(Target, S, T) -> ok

Types:

Target = enum()

S = float()

T = float()

See *multiTexCoord1d/2*

multiTexCoord2fv(Target::enum(), V) -> ok

Types:

V = {S::float(), T::float()}

Equivalent to *multiTexCoord2f(Target, S, T)*.

multiTexCoord2i(Target, S, T) -> ok

Types:

Target = enum()
S = integer()
T = integer()

See *multiTexCoord1d/2*

multiTexCoord2iv(Target::enum(), V) -> ok

Types:

V = {S::integer(), T::integer()}

Equivalent to *multiTexCoord2i(Target, S, T)*.

multiTexCoord2s(Target, S, T) -> ok

Types:

Target = enum()
S = integer()
T = integer()

See *multiTexCoord1d/2*

multiTexCoord2sv(Target::enum(), V) -> ok

Types:

V = {S::integer(), T::integer()}

Equivalent to *multiTexCoord2s(Target, S, T)*.

multiTexCoord3d(Target, S, T, R) -> ok

Types:

Target = enum()
S = float()
T = float()
R = float()

See *multiTexCoord1d/2*

multiTexCoord3dv(Target::enum(), V) -> ok

Types:

V = {S::float(), T::float(), R::float()}

Equivalent to *multiTexCoord3d(Target, S, T, R)*.

multiTexCoord3f(Target, S, T, R) -> ok

Types:

Target = enum()
S = float()
T = float()
R = float()

See *multiTexCoord1d/2*

multiTexCoord3fv(Target::enum(), V) -> ok

Types:

V = {S::float(), T::float(), R::float()}

Equivalent to *multiTexCoord3f(Target, S, T, R)*.

multiTexCoord3i(Target, S, T, R) -> ok

Types:

Target = enum()
S = integer()
T = integer()
R = integer()

See *multiTexCoord1d/2*

multiTexCoord3iv(Target::enum(), V) -> ok

Types:

V = {S::integer(), T::integer(), R::integer()}

Equivalent to *multiTexCoord3i(Target, S, T, R)*.

multiTexCoord3s(Target, S, T, R) -> ok

Types:

Target = enum()
S = integer()
T = integer()
R = integer()

See *multiTexCoord1d/2*

multiTexCoord3sv(Target::enum(), V) -> ok

Types:

V = {S::integer(), T::integer(), R::integer()}

Equivalent to *multiTexCoord3s(Target, S, T, R)*.

multiTexCoord4d(Target, S, T, R, Q) -> ok

Types:

Target = enum()
S = float()

```
T = float()
R = float()
Q = float()
```

See *multiTexCoord1d/2*

multiTexCoord4dv(Target::enum(), V) -> ok

Types:

```
V = {S::float(), T::float(), R::float(), Q::float()}
```

Equivalent to *multiTexCoord4d(Target, S, T, R, Q)*.

multiTexCoord4f(Target, S, T, R, Q) -> ok

Types:

```
Target = enum()
S = float()
T = float()
R = float()
Q = float()
```

See *multiTexCoord1d/2*

multiTexCoord4fv(Target::enum(), V) -> ok

Types:

```
V = {S::float(), T::float(), R::float(), Q::float()}
```

Equivalent to *multiTexCoord4f(Target, S, T, R, Q)*.

multiTexCoord4i(Target, S, T, R, Q) -> ok

Types:

```
Target = enum()
S = integer()
T = integer()
R = integer()
Q = integer()
```

See *multiTexCoord1d/2*

multiTexCoord4iv(Target::enum(), V) -> ok

Types:

```
V = {S::integer(), T::integer(), R::integer(), Q::integer()}
```

Equivalent to *multiTexCoord4i(Target, S, T, R, Q)*.

multiTexCoord4s(Target, S, T, R, Q) -> ok

Types:

```
Target = enum()
S = integer()
T = integer()
```

```
R = integer()
Q = integer()
```

See *multiTexCoord1d/2*

multiTexCoord4sv(Target::enum(), V) -> ok

Types:

```
V = {S::integer(), T::integer(), R::integer(), Q::integer()}
```

Equivalent to *multiTexCoord4s(Target, S, T, R, Q)*.

loadTransposeMatrixf(M) -> ok

Types:

```
M = matrix()
```

Replace the current matrix with the specified row-major ordered matrix

`gl:loadTransposeMatrix` replaces the current matrix with the one whose elements are specified by `M` . The current matrix is the projection matrix, modelview matrix, or texture matrix, depending on the current matrix mode (see *gl:matrixMode/1*).

The current matrix, M, defines a transformation of coordinates. For instance, assume M refers to the modelview matrix. If v=(v[0] v[1] v[2] v[3]) is the set of object coordinates of a vertex, and `M` points to an array of 16 single- or double-precision floating-point values m={m[0] m[1] ... m[15]}, then the modelview transformation M(v) does the following:

M(v)=(m[0] m[1] m[2] m[3] m[4] m[5] m[6] m[7] m[8] m[9] m[10] m[11] m[12] m[13] m[14] m[15])*(v[0] v[1] v[2] v[3])

Projection and texture transformations are similarly defined.

Calling `gl:loadTransposeMatrix` with matrix M is identical in operation to *gl:loadMatrixd/1* with M T, where T represents the transpose.

See **external** documentation.

loadTransposeMatrixd(M) -> ok

Types:

```
M = matrix()
```

See *loadTransposeMatrixf/1*

multTransposeMatrixf(M) -> ok

Types:

```
M = matrix()
```

Multiply the current matrix with the specified row-major ordered matrix

`gl:multTransposeMatrix` multiplies the current matrix with the one specified using `M` , and replaces the current matrix with the product.

The current matrix is determined by the current matrix mode (see *gl:matrixMode/1*). It is either the projection matrix, modelview matrix, or the texture matrix.

See **external** documentation.

multTransposeMatrixd(M) -> ok

Types:

M = matrix()

See *multTransposeMatrixf/1*

blendFuncSeparate(SfactorRGB, DfactorRGB, SfactorAlpha, DfactorAlpha) -> ok

Types:

SfactorRGB = enum()
DfactorRGB = enum()
SfactorAlpha = enum()
DfactorAlpha = enum()

Specify pixel arithmetic for RGB and alpha components separately

Pixels can be drawn using a function that blends the incoming (source) RGBA values with the RGBA values that are already in the frame buffer (the destination values). Blending is initially disabled. Use *gl:enable/1* and *gl:enable/1* with argument `?GL_BLEND` to enable and disable blending.

`gl:blendFuncSeparate` defines the operation of blending for all draw buffers when it is enabled. `gl:blendFuncSeparatei` defines the operation of blending for a single draw buffer specified by `Buf` when enabled for that draw buffer. `SrcRGB` specifies which method is used to scale the source RGB-color components. `DstRGB` specifies which method is used to scale the destination RGB-color components. Likewise, `SrcAlpha` specifies which method is used to scale the source alpha color component, and `DstAlpha` specifies which method is used to scale the destination alpha component. The possible methods are described in the following table. Each method defines four scale factors, one each for red, green, blue, and alpha.

In the table and in subsequent equations, first source, second source and destination color components are referred to as (R s0 G s0 B s0 A s0), (R s1 G s1 B s1 A s1), and (R d G d B d A d), respectively. The color specified by *gl:blendColor/4* is referred to as (R c G c B c A c). They are understood to have integer values between 0 and (k R k G k B k A), where

k c= 2(m c)-1

and (m R m G m B m A) is the number of red, green, blue, and alpha bitplanes.

Source and destination scale factors are referred to as (s R s G s B s A) and (d R d G d B d A). All scale factors have range [0 1].

`ParameterRGB FactorAlpha Factor`
`?GL_ZERO`(0 0 0) 0
`?GL_ONE` (1 1 1) 1
`?GL_SRC_COLOR`(R s0 k/R G s0 k/G B s0 k/B) A s0 k/A
`?GL_ONE_MINUS_SRC_COLOR`(1 1 1 1)-(R s0 k/R G s0 k/G B s0 k/B) 1-A s0 k/A
`?GL_DST_COLOR`(R d k/R G d k/G B d k/B) A d k/A
`?GL_ONE_MINUS_DST_COLOR` (1 1 1)-(R d k/R G d k/G B d k/B) 1-A d k/A
`?GL_SRC_ALPHA`(A s0 k/A A s0 k/A A s0 k/A) A s0 k/A
`?GL_ONE_MINUS_SRC_ALPHA`(1 1 1)-(A s0 k/A A s0 k/A A s0 k/A) 1-A s0 k/A
`?GL_DST_ALPHA`(A d k/A A d k/A A d k/A) A d k/A
`?GL_ONE_MINUS_DST_ALPHA` (1 1 1)-(A d k/A A d k/A A d k/A) 1-A d k/A
`?GL_CONSTANT_COLOR`(R c G c B c) A c
`?GL_ONE_MINUS_CONSTANT_COLOR`(1 1 1)-(R c G c B c) 1-A c
`?GL_CONSTANT_ALPHA`(A c A c A c) A c
`?GL_ONE_MINUS_CONSTANT_ALPHA` (1 1 1)-(A c A c A c) 1-A c
`?GL_SRC_ALPHA_SATURATE`(i i i) 1
`?GL_SRC1_COLOR`(R s1 k/R G s1 k/G B s1 k/B) A s1 k/A
`?GL_ONE_MINUS_SRC_COLOR` (1 1 1 1)-(R s1 k/R G s1 k/G B s1 k/B) 1-A s1 k/A
`?GL_SRC1_ALPHA`(A s1 k/A A s1 k/A A s1 k/A) A s1 k/A

?GL_ONE_MINUS_SRC_ALPHA(1 1 1)-(A s1 k/A A s1 k/A A s1 k/A) 1-A s1 k/A

In the table,

i= min(A s 1-(A d))

To determine the blended RGBA values of a pixel, the system uses the following equations:

R d= min(k R R s s R+R d d R) G d= min(k G G s s G+G d d G) B d= min(k B B s s B+B d d B) A d= min(k A A s s A+A d d A)

Despite the apparent precision of the above equations, blending arithmetic is not exactly specified, because blending operates with imprecise integer color values. However, a blend factor that should be equal to 1 is guaranteed not to modify its multiplicand, and a blend factor equal to 0 reduces its multiplicand to 0. For example, when SrcRGB is ?GL_SRC_ALPHA , DstRGB is ?GL_ONE_MINUS_SRC_ALPHA, and A s is equal to k A, the equations reduce to simple replacement:

R d= R s G d= G s B d= B s A d= A s

See **external** documentation.

multiDrawArrays(Mode, First, Count) -> ok

Types:

```
Mode = enum()
First = [integer()]
Count = [integer()]
```

Render multiple sets of primitives from array data

gl:multiDrawArrays specifies multiple sets of geometric primitives with very few subroutine calls. Instead of calling a GL procedure to pass each individual vertex, normal, texture coordinate, edge flag, or color, you can prespecify separate arrays of vertices, normals, and colors and use them to construct a sequence of primitives with a single call to gl:multiDrawArrays.

gl:multiDrawArrays behaves identically to *gl:drawArrays/3* except that Primcount separate ranges of elements are specified instead.

When gl:multiDrawArrays is called, it uses Count sequential elements from each enabled array to construct a sequence of geometric primitives, beginning with element First . Mode specifies what kind of primitives are constructed, and how the array elements construct those primitives.

Vertex attributes that are modified by gl:multiDrawArrays have an unspecified value after gl:multiDrawArrays returns. Attributes that aren't modified remain well defined.

See **external** documentation.

pointParameterf(Pname, Param) -> ok

Types:

```
Pname = enum()
Param = float()
```

Specify point parameters

The following values are accepted for Pname :

?GL_POINT_FADE_THRESHOLD_SIZE: Params is a single floating-point value that specifies the threshold value to which point sizes are clamped if they exceed the specified value. The default value is 1.0.

?GL_POINT_SPRITE_COORD_ORIGIN: Params is a single enum specifying the point sprite texture coordinate origin, either ?GL_LOWER_LEFT or ?GL_UPPER_LEFT. The default value is ?GL_UPPER_LEFT.

See **external** documentation.

pointParameterfv(Pname, Params) -> ok

Types:

Pname = enum()

Params = {float()}

See *pointParameterf/2*

pointParameteri(Pname, Param) -> ok

Types:

Pname = enum()

Param = integer()

See *pointParameterf/2*

pointParameteriv(Pname, Params) -> ok

Types:

Pname = enum()

Params = {integer()}

See *pointParameterf/2*

fogCoordf(Coord) -> ok

Types:

Coord = float()

Set the current fog coordinates

`gl:fogCoord` specifies the fog coordinate that is associated with each vertex and the current raster position. The value specified is interpolated and used in computing the fog color (see *gl:fogf/2*).

See **external** documentation.

fogCoordfv(Coord) -> ok

Types:

Coord = {Coord::float()}

Equivalent to *fogCoordf(Coord)*.

fogCoordd(Coord) -> ok

Types:

Coord = float()

See *fogCoordf/1*

fogCoorddv(Coord) -> ok

Types:

Coord = {Coord::float()}

Equivalent to *fogCoordd(Coord)*.

fogCoordPointer(Type, Stride, Pointer) -> ok

Types:

Type = enum()
Stride = integer()
Pointer = offset() | mem()

Define an array of fog coordinates

`gl:fogCoordPointer` specifies the location and data format of an array of fog coordinates to use when rendering. `Type` specifies the data type of each fog coordinate, and `Stride` specifies the byte stride from one fog coordinate to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays.

If a non-zero named buffer object is bound to the `?GL_ARRAY_BUFFER` target (see *gl:bindBuffer/2*) while a fog coordinate array is specified, `Pointer` is treated as a byte offset into the buffer object's data store. Also, the buffer object binding (`?GL_ARRAY_BUFFER_BINDING`) is saved as fog coordinate vertex array client-side state (`?GL_FOG_COORD_ARRAY_BUFFER_BINDING`).

When a fog coordinate array is specified, `Type` , `Stride` , and `Pointer` are saved as client-side state, in addition to the current vertex array buffer object binding.

To enable and disable the fog coordinate array, call *gl:enableClientState/1* and *gl:enableClientState/1* with the argument `?GL_FOG_COORD_ARRAY`. If enabled, the fog coordinate array is used when *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see `glMultiDrawElements` , *gl:drawRangeElements/6* , or *gl:arrayElement/1* is called.

See **external** documentation.

secondaryColor3b(Red, Green, Blue) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()

Set the current secondary color

The GL stores both a primary four-valued RGBA color and a secondary four-valued RGBA color (where alpha is always set to 0.0) that is associated with every vertex.

The secondary color is interpolated and applied to each fragment during rasterization when `?GL_COLOR_SUM` is enabled. When lighting is enabled, and `?GL_SEPARATE_SPECULAR_COLOR` is specified, the value of the secondary color is assigned the value computed from the specular term of the lighting computation. Both the primary and secondary current colors are applied to each fragment, regardless of the state of `?GL_COLOR_SUM`, under such conditions. When `?GL_SEPARATE_SPECULAR_COLOR` is specified, the value returned from querying the current secondary color is undefined.

`gl:secondaryColor3b`, `gl:secondaryColor3s`, and `gl:secondaryColor3i` take three signed byte, short, or long integers as arguments. When `v` is appended to the name, the color commands can take a pointer to an array of such values.

Color values are stored in floating-point format, with unspecified mantissa and exponent sizes. Unsigned integer color components, when specified, are linearly mapped to floating-point values such that the largest representable value maps to 1.0 (full intensity), and 0 maps to 0.0 (zero intensity). Signed integer color components, when specified, are linearly mapped to floating-point values such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. (Note that this mapping does not convert 0 precisely to 0.0). Floating-point values are mapped directly.

Neither floating-point nor signed integer values are clamped to the range [0 1] before the current color is updated. However, color components are clamped to this range before they are interpolated or written into a color buffer.

See **external** documentation.

secondaryColor3bv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *secondaryColor3b(Red, Green, Blue)*.

secondaryColor3d(Red, Green, Blue) -> ok

Types:

Red = float()
Green = float()
Blue = float()

See *secondaryColor3b/3*

secondaryColor3dv(V) -> ok

Types:

V = {Red::float(), Green::float(), Blue::float()}

Equivalent to *secondaryColor3d(Red, Green, Blue)*.

secondaryColor3f(Red, Green, Blue) -> ok

Types:

Red = float()
Green = float()
Blue = float()

See *secondaryColor3b/3*

secondaryColor3fv(V) -> ok

Types:

V = {Red::float(), Green::float(), Blue::float()}

Equivalent to *secondaryColor3f(Red, Green, Blue)*.

secondaryColor3i(Red, Green, Blue) -> ok

Types:

Red = integer()
Green = integer()
Blue = integer()

See *secondaryColor3b/3*

secondaryColor3iv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *secondaryColor3i(Red, Green, Blue)*.

secondaryColor3s(Red, Green, Blue) -> ok

Types:

```
Red = integer()
Green = integer()
Blue = integer()
```

See *secondaryColor3b/3*

secondaryColor3sv(V) -> ok

Types:

```
V = {Red::integer(), Green::integer(), Blue::integer()}
```

Equivalent to *secondaryColor3s(Red, Green, Blue)*.

secondaryColor3ub(Red, Green, Blue) -> ok

Types:

```
Red = integer()
Green = integer()
Blue = integer()
```

See *secondaryColor3b/3*

secondaryColor3ubv(V) -> ok

Types:

```
V = {Red::integer(), Green::integer(), Blue::integer()}
```

Equivalent to *secondaryColor3ub(Red, Green, Blue)*.

secondaryColor3ui(Red, Green, Blue) -> ok

Types:

```
Red = integer()
Green = integer()
Blue = integer()
```

See *secondaryColor3b/3*

secondaryColor3uiv(V) -> ok

Types:

```
V = {Red::integer(), Green::integer(), Blue::integer()}
```

Equivalent to *secondaryColor3ui(Red, Green, Blue)*.

secondaryColor3us(Red, Green, Blue) -> ok

Types:

```
Red = integer()
Green = integer()
Blue = integer()
```

See *secondaryColor3b/3*

secondaryColor3usv(V) -> ok

Types:

V = {Red::integer(), Green::integer(), Blue::integer()}

Equivalent to *secondaryColor3us(Red, Green, Blue)*.

secondaryColorPointer(Size, Type, Stride, Pointer) -> ok

Types:

Size = integer()
Type = enum()
Stride = integer()
Pointer = offset() | mem()

Define an array of secondary colors

`gl:secondaryColorPointer` specifies the location and data format of an array of color components to use when rendering. `Size` specifies the number of components per color, and must be 3. `Type` specifies the data type of each color component, and `Stride` specifies the byte stride from one color to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays.

If a non-zero named buffer object is bound to the `?GL_ARRAY_BUFFER` target (see *gl:bindBuffer/2*) while a secondary color array is specified, `Pointer` is treated as a byte offset into the buffer object's data store. Also, the buffer object binding (`?GL_ARRAY_BUFFER_BINDING`) is saved as secondary color vertex array client-side state (`?GL_SECONDARY_COLOR_ARRAY_BUFFER_BINDING`).

When a secondary color array is specified, `Size` , `Type` , `Stride` , and `Pointer` are saved as client-side state, in addition to the current vertex array buffer object binding.

To enable and disable the secondary color array, call *gl:enableClientState/1* and *gl:enableClientState/1* with the argument `?GL_SECONDARY_COLOR_ARRAY`. If enabled, the secondary color array is used when *gl:arrayElement/1* , *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see `glMultiDrawElements`, or *gl:drawRangeElements/6* is called.

See **external** documentation.

windowPos2d(X, Y) -> ok

Types:

X = float()
Y = float()

Specify the raster position in window coordinates for pixel operations

The GL maintains a 3D position in window coordinates. This position, called the raster position, is used to position pixel and bitmap write operations. It is maintained with subpixel accuracy. See *gl:bitmap/7* , *gl:drawPixels/5* , and *gl:copyPixels/5* .

`gl:windowPos2` specifies the x and y coordinates, while z is implicitly set to 0. `gl:windowPos3` specifies all three coordinates. The w coordinate of the current raster position is always set to 1.0.

`gl:windowPos` directly updates the x and y coordinates of the current raster position with the values specified. That is, the values are neither transformed by the current modelview and projection matrices, nor by the viewport-to-window transform. The z coordinate of the current raster position is updated in the following manner:

z={n f(n+z*(f-n)) if z<= 0 if z>= 1(otherwise))

where n is ?GL_DEPTH_RANGE's near value, and f is ?GL_DEPTH_RANGE's far value. See *gl:depthRange/2* .

The specified coordinates are not clip-tested, causing the raster position to always be valid.

The current raster position also includes some associated color data and texture coordinates. If lighting is enabled, then ?GL_CURRENT_RASTER_COLOR (in RGBA mode) or ?GL_CURRENT_RASTER_INDEX (in color index mode) is set to the color produced by the lighting calculation (see *gl:lightf/3* , *gl:lightModelf/2* , and *gl:shadeModel/1*). If lighting is disabled, current color (in RGBA mode, state variable ?GL_CURRENT_COLOR) or color index (in color index mode, state variable ?GL_CURRENT_INDEX) is used to update the current raster color. ?GL_CURRENT_RASTER_SECONDARY_COLOR (in RGBA mode) is likewise updated.

Likewise, ?GL_CURRENT_RASTER_TEXTURE_COORDS is updated as a function of ?GL_CURRENT_TEXTURE_COORDS , based on the texture matrix and the texture generation functions (see *gl:texGend/3*). The ?GL_CURRENT_RASTER_DISTANCE is set to the ?GL_CURRENT_FOG_COORD.

See **external** documentation.

windowPos2dv(V) -> ok

Types:

V = {X::float(), Y::float()}

Equivalent to *windowPos2d(X, Y)*.

windowPos2f(X, Y) -> ok

Types:

X = float()
Y = float()

See *windowPos2d/2*

windowPos2fv(V) -> ok

Types:

V = {X::float(), Y::float()}

Equivalent to *windowPos2f(X, Y)*.

windowPos2i(X, Y) -> ok

Types:

X = integer()
Y = integer()

See *windowPos2d/2*

windowPos2iv(V) -> ok

Types:

V = {X::integer(), Y::integer()}

Equivalent to *windowPos2i(X, Y)*.

windowPos2s(X, Y) -> ok

Types:

X = integer()
Y = integer()

See *windowPos2d/2*

windowPos2sv(V) -> ok

Types:

V = {X::integer(), Y::integer()}

Equivalent to *windowPos2s(X, Y)*.

windowPos3d(X, Y, Z) -> ok

Types:

X = float()
Y = float()
Z = float()

See *windowPos2d/2*

windowPos3dv(V) -> ok

Types:

V = {X::float(), Y::float(), Z::float()}

Equivalent to *windowPos3d(X, Y, Z)*.

windowPos3f(X, Y, Z) -> ok

Types:

X = float()
Y = float()
Z = float()

See *windowPos2d/2*

windowPos3fv(V) -> ok

Types:

V = {X::float(), Y::float(), Z::float()}

Equivalent to *windowPos3f(X, Y, Z)*.

windowPos3i(X, Y, Z) -> ok

Types:

X = integer()
Y = integer()
Z = integer()

See *windowPos2d/2*

windowPos3iv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer()}

Equivalent to *windowPos3i(X, Y, Z)*.

windowPos3s(X, Y, Z) -> ok

Types:

X = integer()
Y = integer()
Z = integer()

See *windowPos2d/2*

windowPos3sv(V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer()}

Equivalent to *windowPos3s(X, Y, Z)*.

genQueries(N) -> [integer()]

Types:

N = integer()

Generate query object names

`gl:genQueries` returns `N` query object names in `Ids` . There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to `gl:genQueries`.

Query object names returned by a call to `gl:genQueries` are not returned by subsequent calls, unless they are first deleted with *gl:deleteQueries/1* .

No query objects are associated with the returned query object names until they are first used by calling *gl:beginQuery/2* .

See **external** documentation.

deleteQueries(Ids) -> ok

Types:

Ids = [integer()]

Delete named query objects

`gl:deleteQueries` deletes `N` query objects named by the elements of the array `Ids` . After a query object is deleted, it has no contents, and its name is free for reuse (for example by *gl:genQueries/1*).

`gl:deleteQueries` silently ignores 0's and names that do not correspond to existing query objects.

See **external** documentation.

isQuery(Id) -> 0 | 1

Types:

Id = integer()

Determine if a name corresponds to a query object

`gl:isQuery` returns `?GL_TRUE` if `Id` is currently the name of a query object. If `Id` is zero, or is a non-zero value that is not currently the name of a query object, or if an error occurs, `gl:isQuery` returns `?GL_FALSE`.

A name returned by *gl:genQueries/1* , but not yet associated with a query object by calling *gl:beginQuery/2* , is not the name of a query object.

See **external** documentation.

beginQuery(Target, Id) -> ok

Types:

Target = enum()

Id = integer()

Delimit the boundaries of a query object

`gl:beginQuery` and *gl:beginQuery/2* delimit the boundaries of a query object. `Query` must be a name previously returned from a call to *gl:genQueries/1* . If a query object with name `Id` does not yet exist it is created with the type determined by `Target` . `Target` must be one of `?GL_SAMPLES_PASSED`, `?GL_ANY_SAMPLES_PASSED`, `?GL_PRIMITIVES_GENERATED` , `?GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN`, or `?GL_TIME_ELAPSED`. The behavior of the query object depends on its type and is as follows.

If `Target` is `?GL_SAMPLES_PASSED`, `Id` must be an unused name, or the name of an existing occlusion query object. When `gl:beginQuery` is executed, the query object's samples-passed counter is reset to 0. Subsequent rendering will increment the counter for every sample that passes the depth test. If the value of `?GL_SAMPLE_BUFFERS` is 0, then the samples-passed count is incremented by 1 for each fragment. If the value of `?GL_SAMPLE_BUFFERS` is 1, then the samples-passed count is incremented by the number of samples whose coverage bit is set. However, implementations, at their discression may instead increase the samples-passed count by the value of `?GL_SAMPLES` if any sample in the fragment is covered. When `gl:endQuery` is executed, the samples-passed counter is assigned to the query object's result value. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`.

If `Target` is `?GL_ANY_SAMPLES_PASSED`, `Id` must be an unused name, or the name of an existing boolean occlusion query object. When `gl:beginQuery` is executed, the query object's samples-passed flag is reset to `?GL_FALSE`. Subsequent rendering causes the flag to be set to `?GL_TRUE` if any sample passes the depth test. When `gl:endQuery` is executed, the samples-passed flag is assigned to the query object's result value. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`.

If `Target` is `?GL_PRIMITIVES_GENERATED`, `Id` must be an unused name, or the name of an existing primitive query object previously bound to the `?GL_PRIMITIVES_GENERATED` query binding. When `gl:beginQuery` is executed, the query object's primitives-generated counter is reset to 0. Subsequent rendering will increment the counter once for every vertex that is emitted from the geometry shader, or from the vertex shader if no geometry shader is present. When `gl:endQuery` is executed, the primitives-generated counter is assigned to the query object's result value. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`.

If `Target` is `?GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN`, `Id` must be an unused name, or the name of an existing primitive query object previously bound to the `?GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN` query binding. When `gl:beginQuery` is executed, the query object's primitives-written counter is reset to 0. Subsequent rendering will increment the counter once for every vertex that is written into the bound transform feedback buffer(s). If transform feedback mode is not activated between the call to `gl:beginQuery` and `gl:endQuery`, the counter will not be incremented. When `gl:endQuery` is executed, the primitives-written counter is assigned to the query object's result value. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`.

If `Target` is `?GL_TIME_ELAPSED`, `Id` must be an unused name, or the name of an existing timer query object previously bound to the `?GL_TIME_ELAPSED` query binding. When `gl:beginQuery` is executed, the query object's time counter is reset to 0. When `gl:endQuery` is executed, the elapsed server time that has passed since the call to `gl:beginQuery` is written into the query object's time counter. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT` .

Querying the ?GL_QUERY_RESULT implicitly flushes the GL pipeline until the rendering delimited by the query object has completed and the result is available. ?GL_QUERY_RESULT_AVAILABLE can be queried to determine if the result is immediately available or if the rendering is not yet complete.

See **external** documentation.

endQuery(Target) -> ok

Types:

Target = enum()

See *beginQuery/2*

getQueryiv(Target, Pname) -> integer()

Types:

Target = enum()

Pname = enum()

glGetQuery

See **external** documentation.

getQueryObjectiv(Id, Pname) -> integer()

Types:

Id = integer()

Pname = enum()

Return parameters of a query object

gl:getQueryObject returns in Params a selected parameter of the query object specified by Id .

Pname names a specific query object parameter. Pname can be as follows:

?GL_QUERY_RESULT: Params returns the value of the query object's passed samples counter. The initial value is 0.

?GL_QUERY_RESULT_AVAILABLE: Params returns whether the passed samples counter is immediately available. If a delay would occur waiting for the query result, ?GL_FALSE is returned. Otherwise, ?GL_TRUE is returned, which also indicates that the results of all previous queries are available as well.

See **external** documentation.

getQueryObjectuiv(Id, Pname) -> integer()

Types:

Id = integer()

Pname = enum()

See *getQueryObjectiv/2*

bindBuffer(Target, Buffer) -> ok

Types:

Target = enum()

Buffer = integer()

Bind a named buffer object

`gl:bindBuffer` binds a buffer object to the specified buffer binding point. Calling `gl:bindBuffer` with `Target` set to one of the accepted symbolic constants and `Buffer` set to the name of a buffer object binds that buffer object name to the target. If no buffer object with name `Buffer` exists, one is created with that name. When a buffer object is bound to a target, the previous binding for that target is automatically broken.

Buffer object names are unsigned integers. The value zero is reserved, but there is no default buffer object for each buffer object target. Instead, `Buffer` set to zero effectively unbinds any buffer object previously bound, and restores client memory usage for that buffer object target (if supported for that target). Buffer object names and the corresponding buffer object contents are local to the shared object space of the current GL rendering context; two rendering contexts share buffer object names only if they explicitly enable sharing between contexts through the appropriate GL windows interfaces functions.

gl:genBuffers/1 must be used to generate a set of unused buffer object names.

The state of a buffer object immediately after it is first bound is an unmapped zero-sized memory buffer with `?GL_READ_WRITE` access and `?GL_STATIC_DRAW` usage.

While a non-zero buffer object name is bound, GL operations on the target to which it is bound affect the bound buffer object, and queries of the target to which it is bound return state from the bound buffer object. While buffer object name zero is bound, as in the initial state, attempts to modify or query state on the target to which it is bound generates an `?GL_INVALID_OPERATION` error.

When a non-zero buffer object is bound to the `?GL_ARRAY_BUFFER` target, the vertex array pointer parameter is interpreted as an offset within the buffer object measured in basic machine units.

When a non-zero buffer object is bound to the `?GL_DRAW_INDIRECT_BUFFER` target, parameters for draws issued through *gl:drawArraysIndirect/2* and *gl:drawElementsIndirect/3* are sourced from that buffer object.

While a non-zero buffer object is bound to the `?GL_ELEMENT_ARRAY_BUFFER` target, the indices parameter of *gl:drawElements/4* , *gl:drawElementsInstanced/5* , *gl:drawElementsBaseVertex/5* , *gl:drawRangeElements/6* , *gl:drawRangeElementsBaseVertex/7* , see `glMultiDrawElements` , or see `glMultiDrawElementsBaseVertex` is interpreted as an offset within the buffer object measured in basic machine units.

While a non-zero buffer object is bound to the `?GL_PIXEL_PACK_BUFFER` target, the following commands are affected: *gl:getCompressedTexImage/3* , *gl:getTexImage/5* , and *gl:readPixels/7* . The pointer parameter is interpreted as an offset within the buffer object measured in basic machine units.

While a non-zero buffer object is bound to the `?GL_PIXEL_UNPACK_BUFFER` target, the following commands are affected: *gl:compressedTexImage1D/7* , *gl:compressedTexImage2D/8* , *gl:compressedTexImage3D/9* , *gl:compressedTexSubImage1D/7* , *gl:compressedTexSubImage2D/9* , *gl:compressedTexSubImage3D/11* , *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:texSubImage1D/7* , *gl:texSubImage1D/7* , and *gl:texSubImage1D/7* . The pointer parameter is interpreted as an offset within the buffer object measured in basic machine units.

The buffer targets `?GL_COPY_READ_BUFFER` and `?GL_COPY_WRITE_BUFFER` are provided to allow *gl:copyBufferSubData/5* to be used without disturbing the state of other bindings. However, *gl:copyBufferSubData/5* may be used with any pair of buffer binding points.

The `?GL_TRANSFORM_FEEDBACK_BUFFER` buffer binding point may be passed to `gl:bindBuffer` , but will not directly affect transform feedback state. Instead, the indexed `?GL_TRANSFORM_FEEDBACK_BUFFER` bindings must be used through a call to *gl:bindBufferBase/3* or *gl:bindBufferRange/5* . This will affect the generic `?GL_TRANSFORM_FEEDABCK_BUFFER` binding.

Likewise, the `?GL_UNIFORM_BUFFER` and `?GL_ATOMIC_COUNTER_BUFFER` buffer binding points may be used, but do not directly affect uniform buffer or atomic counter buffer state, respectively. *gl:bindBufferBase/3* or *gl:bindBufferRange/5* must be used to bind a buffer to an indexed uniform buffer or atomic counter buffer binding point.

A buffer object binding created with `gl:bindBuffer` remains active until a different buffer object name is bound to the same target, or until the bound buffer object is deleted with *gl:deleteBuffers/1* .

Once created, a named buffer object may be re-bound to any target as often as needed. However, the GL implementation may make choices about how to optimize the storage of a buffer object based on its initial binding target.

See **external** documentation.

deleteBuffers(Buffers) -> ok

Types:

Buffers = [integer()]

Delete named buffer objects

`gl:deleteBuffers` deletes `N` buffer objects named by the elements of the array `Buffers` . After a buffer object is deleted, it has no contents, and its name is free for reuse (for example by *gl:genBuffers/1*). If a buffer object that is currently bound is deleted, the binding reverts to 0 (the absence of any buffer object).

`gl:deleteBuffers` silently ignores 0's and names that do not correspond to existing buffer objects.

See **external** documentation.

genBuffers(N) -> [integer()]

Types:

N = integer()

Generate buffer object names

`gl:genBuffers` returns `N` buffer object names in `Buffers` . There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to `gl:genBuffers` .

Buffer object names returned by a call to `gl:genBuffers` are not returned by subsequent calls, unless they are first deleted with *gl:deleteBuffers/1* .

No buffer objects are associated with the returned buffer object names until they are first bound by calling *gl:bindBuffer/2* .

See **external** documentation.

isBuffer(Buffer) -> 0 | 1

Types:

Buffer = integer()

Determine if a name corresponds to a buffer object

`gl:isBuffer` returns `?GL_TRUE` if `Buffer` is currently the name of a buffer object. If `Buffer` is zero, or is a non-zero value that is not currently the name of a buffer object, or if an error occurs, `gl:isBuffer` returns `?GL_FALSE` .

A name returned by *gl:genBuffers/1* , but not yet associated with a buffer object by calling *gl:bindBuffer/2* , is not the name of a buffer object.

See **external** documentation.

bufferData(Target, Size, Data, Usage) -> ok

Types:

Target = enum()
Size = integer()

```
Data = offset() | mem()
Usage = enum()
```

Creates and initializes a buffer object's data store

`gl:bufferData` creates a new data store for the buffer object currently bound to `Target` . Any pre-existing data store is deleted. The new data store is created with the specified `Size` in bytes and `Usage` . If `Data` is not `?NULL`, the data store is initialized with data from this pointer. In its initial state, the new data store is not mapped, it has a `?NULL` mapped pointer, and its mapped access is `?GL_READ_WRITE` .

`Usage` is a hint to the GL implementation as to how a buffer object's data store will be accessed. This enables the GL implementation to make more intelligent decisions that may significantly impact buffer object performance. It does not, however, constrain the actual usage of the data store. `Usage` can be broken down into two parts: first, the frequency of access (modification and usage), and second, the nature of that access. The frequency of access may be one of these:

STREAM: The data store contents will be modified once and used at most a few times.

STATIC: The data store contents will be modified once and used many times.

DYNAMIC: The data store contents will be modified repeatedly and used many times.

The nature of access may be one of these:

DRAW: The data store contents are modified by the application, and used as the source for GL drawing and image specification commands.

READ: The data store contents are modified by reading data from the GL, and used to return that data when queried by the application.

COPY: The data store contents are modified by reading data from the GL, and used as the source for GL drawing and image specification commands.

See **external** documentation.

bufferSubData(Target, Offset, Size, Data) -> ok

Types:

```
Target = enum()
Offset = integer()
Size = integer()
Data = offset() | mem()
```

Updates a subset of a buffer object's data store

`gl:bufferSubData` redefines some or all of the data store for the buffer object currently bound to `Target` . Data starting at byte offset `Offset` and extending for `Size` bytes is copied to the data store from the memory pointed to by `Data` . An error is thrown if `Offset` and `Size` together define a range beyond the bounds of the buffer object's data store.

See **external** documentation.

getBufferSubData(Target, Offset, Size, Data) -> ok

Types:

```
Target = enum()
Offset = integer()
Size = integer()
Data = mem()
```

Returns a subset of a buffer object's data store

`gl:getBufferSubData` returns some or all of the data from the buffer object currently bound to `Target` . Data starting at byte offset `Offset` and extending for `Size` bytes is copied from the data store to the memory pointed to by `Data` . An error is thrown if the buffer object is currently mapped, or if `Offset` and `Size` together define a range beyond the bounds of the buffer object's data store.

See **external** documentation.

`getBufferParameteriv(Target, Pname) -> integer()`

Types:

`Target = enum()`

`Pname = enum()`

Return parameters of a buffer object

`gl:getBufferParameteriv` returns in `Data` a selected parameter of the buffer object specified by `Target` .

`Value` names a specific buffer object parameter, as follows:

`?GL_BUFFER_ACCESS`: `Params` returns the access policy set while mapping the buffer object. The initial value is `?GL_READ_WRITE`.

`?GL_BUFFER_MAPPED`: `Params` returns a flag indicating whether the buffer object is currently mapped. The initial value is `?GL_FALSE`.

`?GL_BUFFER_SIZE`: `Params` returns the size of the buffer object, measured in bytes. The initial value is 0.

`?GL_BUFFER_USAGE`: `Params` returns the buffer object's usage pattern. The initial value is `?GL_STATIC_DRAW`.

See **external** documentation.

`blendEquationSeparate(ModeRGB, ModeAlpha) -> ok`

Types:

`ModeRGB = enum()`

`ModeAlpha = enum()`

Set the RGB blend equation and the alpha blend equation separately

The blend equations determines how a new pixel (the "source" color) is combined with a pixel already in the framebuffer (the "destination" color). These functions specifie one blend equation for the RGB-color components and one blend equation for the alpha component. `gl:blendEquationSeparatei` specifies the blend equations for a single draw buffer whereas `gl:blendEquationSeparate` sets the blend equations for all draw buffers.

The blend equations use the source and destination blend factors specified by either *gl:blendFunc/2* or *gl:blendFuncSeparate/4* . See *gl:blendFunc/2* or *gl:blendFuncSeparate/4* for a description of the various blend factors.

In the equations that follow, source and destination color components are referred to as (R s G s B s A s) and (R d G d B d A d), respectively. The result color is referred to as (R r G r B r A r). The source and destination blend factors are denoted (s R s G s B s A) and (d R d G d B d A), respectively. For these equations all color components are understood to have values in the range [0 1].

`Mode` `RGB Components` `Alpha Component`
`?GL_FUNC_ADD` Rr= R s s R+R d d R Gr= G s s G+G d d G Br= B s s B+B d d B Ar= A s s A+A d d A
`?GL_FUNC_SUBTRACT` Rr= R s s R-R d d R Gr= G s s G-G d d G Br= B s s B-B d d B Ar= A s s A-A d d A
`?GL_FUNC_REVERSE_SUBTRACT` Rr= R d d R-R s s R Gr= G d d G-G s s G Br= B d d B-B s s B Ar= A d d A-A s s A
`?GL_MIN` Rr= min(R s R d) Gr= min(G s G d) Br= min(B s B d) Ar= min(A s A d)
`?GL_MAX` Rr= max(R s R d) Gr= max(G s G d) Br= max(B s B d) Ar= max(A s A d)

The results of these equations are clamped to the range [0 1].

The ?GL_MIN and ?GL_MAX equations are useful for applications that analyze image data (image thresholding against a constant color, for example). The ?GL_FUNC_ADD equation is useful for antialiasing and transparency, among other things.

Initially, both the RGB blend equation and the alpha blend equation are set to ?GL_FUNC_ADD .

See **external** documentation.

drawBuffers(Bufs) -> ok

Types:

```
Bufs = [enum()]
```

Specifies a list of color buffers to be drawn into

gl:drawBuffers defines an array of buffers into which outputs from the fragment shader data will be written. If a fragment shader writes a value to one or more user defined output variables, then the value of each variable will be written into the buffer specified at a location within Bufs corresponding to the location assigned to that user defined output. The draw buffer used for user defined outputs assigned to locations greater than or equal to N is implicitly set to ?GL_NONE and any data written to such an output is discarded.

The symbolic constants contained in Bufs may be any of the following:

?GL_NONE: The fragment shader output value is not written into any color buffer.

?GL_FRONT_LEFT: The fragment shader output value is written into the front left color buffer.

?GL_FRONT_RIGHT: The fragment shader output value is written into the front right color buffer.

?GL_BACK_LEFT: The fragment shader output value is written into the back left color buffer.

?GL_BACK_RIGHT: The fragment shader output value is written into the back right color buffer.

?GL_COLOR_ATTACHMENTn: The fragment shader output value is written into the nth color attachment of the current framebuffer. n may range from 0 to the value of ?GL_MAX_COLOR_ATTACHMENTS.

Except for ?GL_NONE, the preceding symbolic constants may not appear more than once in Bufs . The maximum number of draw buffers supported is implementation dependent and can be queried by calling *gl:getBooleanv/1* with the argument ?GL_MAX_DRAW_BUFFERS .

See **external** documentation.

stencilOpSeparate(Face, Sfail, Dpfail, Dppass) -> ok

Types:

```
Face = enum()
Sfail = enum()
Dpfail = enum()
Dppass = enum()
```

Set front and/or back stencil test actions

Stenciling, like depth-buffering, enables and disables drawing on a per-pixel basis. You draw into the stencil planes using GL drawing primitives, then render geometry and images, using the stencil planes to mask out portions of the screen. Stenciling is typically used in multipass rendering algorithms to achieve special effects, such as decals, outlining, and constructive solid geometry rendering.

The stencil test conditionally eliminates a pixel based on the outcome of a comparison between the value in the stencil buffer and a reference value. To enable and disable the test, call *gl:enable/1* and *gl:enable/1* with argument ?GL_STENCIL_TEST ; to control it, call *gl:stencilFunc/3* or *gl:stencilFuncSeparate/4* .

There can be two separate sets of `Sfail` , `Dpfail` , and `Dppass` parameters; one affects back-facing polygons, and the other affects front-facing polygons as well as other non-polygon primitives. *gl:stencilOp/3* sets both front and back stencil state to the same values, as if *gl:stencilOpSeparate/4* were called with `Face` set to `?GL_FRONT_AND_BACK`.

`gl:stencilOpSeparate` takes three arguments that indicate what happens to the stored stencil value while stenciling is enabled. If the stencil test fails, no change is made to the pixel's color or depth buffers, and `Sfail` specifies what happens to the stencil buffer contents. The following eight actions are possible.

`?GL_KEEP`: Keeps the current value.

`?GL_ZERO`: Sets the stencil buffer value to 0.

`?GL_REPLACE`: Sets the stencil buffer value to `ref`, as specified by *gl:stencilFunc/3* .

`?GL_INCR`: Increments the current stencil buffer value. Clamps to the maximum representable unsigned value.

`?GL_INCR_WRAP`: Increments the current stencil buffer value. Wraps stencil buffer value to zero when incrementing the maximum representable unsigned value.

`?GL_DECR`: Decrements the current stencil buffer value. Clamps to 0.

`?GL_DECR_WRAP`: Decrements the current stencil buffer value. Wraps stencil buffer value to the maximum representable unsigned value when decrementing a stencil buffer value of zero.

`?GL_INVERT`: Bitwise inverts the current stencil buffer value.

Stencil buffer values are treated as unsigned integers. When incremented and decremented, values are clamped to 0 and 2 n-1, where n is the value returned by querying `?GL_STENCIL_BITS` .

The other two arguments to `gl:stencilOpSeparate` specify stencil buffer actions that depend on whether subsequent depth buffer tests succeed (`Dppass`) or fail (`Dpfail`) (see *gl:depthFunc/1*). The actions are specified using the same eight symbolic constants as `Sfail` . Note that `Dpfail` is ignored when there is no depth buffer, or when the depth buffer is not enabled. In these cases, `Sfail` and `Dppass` specify stencil action when the stencil test fails and passes, respectively.

See **external** documentation.

`stencilFuncSeparate(Face, Func, Ref, Mask) -> ok`

Types:

```
Face = enum()
Func = enum()
Ref = integer()
Mask = integer()
```

Set front and/or back function and reference value for stencil testing

Stenciling, like depth-buffering, enables and disables drawing on a per-pixel basis. You draw into the stencil planes using GL drawing primitives, then render geometry and images, using the stencil planes to mask out portions of the screen. Stenciling is typically used in multipass rendering algorithms to achieve special effects, such as decals, outlining, and constructive solid geometry rendering.

The stencil test conditionally eliminates a pixel based on the outcome of a comparison between the reference value and the value in the stencil buffer. To enable and disable the test, call *gl:enable/1* and *gl:enable/1* with argument `?GL_STENCIL_TEST` . To specify actions based on the outcome of the stencil test, call *gl:stencilOp/3* or *gl:stencilOpSeparate/4* .

There can be two separate sets of `Func` , `Ref` , and `Mask` parameters; one affects back-facing polygons, and the other affects front-facing polygons as well as other non-polygon primitives. *gl:stencilFunc/3* sets both front and back stencil state to the same values, as if *gl:stencilFuncSeparate/4* were called with `Face` set to `?GL_FRONT_AND_BACK`.

Func is a symbolic constant that determines the stencil comparison function. It accepts one of eight values, shown in the following list. Ref is an integer reference value that is used in the stencil comparison. It is clamped to the range [0 2 n-1], where n is the number of bitplanes in the stencil buffer. Mask is bitwise ANDed with both the reference value and the stored stencil value, with the ANDed values participating in the comparison.

If stencil represents the value stored in the corresponding stencil buffer location, the following list shows the effect of each comparison function that can be specified by Func . Only if the comparison succeeds is the pixel passed through to the next stage in the rasterization process (see *gl:stencilOp/3*). All tests treat stencil values as unsigned integers in the range [0 2 n-1], where n is the number of bitplanes in the stencil buffer.

The following values are accepted by Func :

?GL_NEVER: Always fails.

?GL_LESS: Passes if (Ref & Mask) < (stencil & Mask).

?GL_LEQUAL: Passes if (Ref & Mask) <= (stencil & Mask).

?GL_GREATER: Passes if (Ref & Mask) > (stencil & Mask).

?GL_GEQUAL: Passes if (Ref & Mask) >= (stencil & Mask).

?GL_EQUAL: Passes if (Ref & Mask) = (stencil & Mask).

?GL_NOTEQUAL: Passes if (Ref & Mask) != (stencil & Mask).

?GL_ALWAYS: Always passes.

See **external** documentation.

stencilMaskSeparate(Face, Mask) -> ok

Types:

```
Face = enum()
Mask = integer()
```

Control the front and/or back writing of individual bits in the stencil planes

gl:stencilMaskSeparate controls the writing of individual bits in the stencil planes. The least significant n bits of Mask , where n is the number of bits in the stencil buffer, specify a mask. Where a 1 appears in the mask, it's possible to write to the corresponding bit in the stencil buffer. Where a 0 appears, the corresponding bit is write-protected. Initially, all bits are enabled for writing.

There can be two separate Mask writemasks; one affects back-facing polygons, and the other affects front-facing polygons as well as other non-polygon primitives. *gl:stencilMask/1* sets both front and back stencil writemasks to the same values, as if *gl:stencilMaskSeparate/2* were called with Face set to ?GL_FRONT_AND_BACK.

See **external** documentation.

attachShader(Program, Shader) -> ok

Types:

```
Program = integer()
Shader = integer()
```

Attaches a shader object to a program object

In order to create a complete shader program, there must be a way to specify the list of things that will be linked together. Program objects provide this mechanism. Shaders that are to be linked together in a program object must first be attached to that program object. gl:attachShader attaches the shader object specified by Shader to the program object specified by Program . This indicates that Shader will be included in link operations that will be performed on Program .

All operations that can be performed on a shader object are valid whether or not the shader object is attached to a program object. It is permissible to attach a shader object to a program object before source code has been loaded into the shader object or before the shader object has been compiled. It is permissible to attach multiple shader objects of the same type because each may contain a portion of the complete shader. It is also permissible to attach a shader object to more than one program object. If a shader object is deleted while it is attached to a program object, it will be flagged for deletion, and deletion will not occur until *gl:detachShader/2* is called to detach it from all program objects to which it is attached.

See **external** documentation.

bindAttribLocation(Program, Index, Name) -> ok

Types:

```
Program = integer()
Index = integer()
Name = string()
```

Associates a generic vertex attribute index with a named attribute variable

`gl:bindAttribLocation` is used to associate a user-defined attribute variable in the program object specified by `Program` with a generic vertex attribute index. The name of the user-defined attribute variable is passed as a null terminated string in `Name` . The generic vertex attribute index to be bound to this variable is specified by `Index` . When `Program` is made part of current state, values provided via the generic vertex attribute `Index` will modify the value of the user-defined attribute variable specified by `Name` .

If `Name` refers to a matrix attribute variable, `Index` refers to the first column of the matrix. Other matrix columns are then automatically bound to locations `Index+1` for a matrix of type `mat2`; `Index+1` and `Index+2` for a matrix of type `mat3`; and `Index+1` , `Index+2` , and `Index+3` for a matrix of type `mat4` .

This command makes it possible for vertex shaders to use descriptive names for attribute variables rather than generic variables that are numbered from 0 to `?GL_MAX_VERTEX_ATTRIBS` -1. The values sent to each generic attribute index are part of current state. If a different program object is made current by calling *gl:useProgram/1* , the generic vertex attributes are tracked in such a way that the same values will be observed by attributes in the new program object that are also bound to `Index` .

Attribute variable name-to-generic attribute index bindings for a program object can be explicitly assigned at any time by calling `gl:bindAttribLocation`. Attribute bindings do not go into effect until *gl:linkProgram/1* is called. After a program object has been linked successfully, the index values for generic attributes remain fixed (and their values can be queried) until the next link command occurs.

Any attribute binding that occurs after the program object has been linked will not take effect until the next time the program object is linked.

See **external** documentation.

compileShader(Shader) -> ok

Types:

```
Shader = integer()
```

Compiles a shader object

`gl:compileShader` compiles the source code strings that have been stored in the shader object specified by `Shader` .

The compilation status will be stored as part of the shader object's state. This value will be set to `?GL_TRUE` if the shader was compiled without errors and is ready for use, and `?GL_FALSE` otherwise. It can be queried by calling *gl:getShaderiv/2* with arguments `Shader` and `?GL_COMPILE_STATUS`.

Compilation of a shader can fail for a number of reasons as specified by the OpenGL Shading Language Specification. Whether or not the compilation was successful, information about the compilation can be obtained from the shader object's information log by calling *gl:getShaderInfoLog/2* .

See **external** documentation.

createProgram() -> integer()

Creates a program object

`gl:createProgram` creates an empty program object and returns a non-zero value by which it can be referenced. A program object is an object to which shader objects can be attached. This provides a mechanism to specify the shader objects that will be linked to create a program. It also provides a means for checking the compatibility of the shaders that will be used to create a program (for instance, checking the compatibility between a vertex shader and a fragment shader). When no longer needed as part of a program object, shader objects can be detached.

One or more executables are created in a program object by successfully attaching shader objects to it with *gl:attachShader/2* , successfully compiling the shader objects with *gl:compileShader/1* , and successfully linking the program object with *gl:linkProgram/1* . These executables are made part of current state when *gl:useProgram/1* is called. Program objects can be deleted by calling *gl:deleteProgram/1* . The memory associated with the program object will be deleted when it is no longer part of current rendering state for any context.

See **external** documentation.

createShader(Type) -> integer()

Types:

Type = enum()

Creates a shader object

`gl:createShader` creates an empty shader object and returns a non-zero value by which it can be referenced. A shader object is used to maintain the source code strings that define a shader. `ShaderType` indicates the type of shader to be created. Five types of shader are supported. A shader of type `?GL_VERTEX_SHADER` is a shader that is intended to run on the programmable vertex processor. A shader of type `?GL_TESS_CONTROL_SHADER` is a shader that is intended to run on the programmable tessellation processor in the control stage. A shader of type `?GL_TESS_EVALUATION_SHADER` is a shader that is intended to run on the programmable tessellation processor in the evaluation stage. A shader of type `?GL_GEOMETRY_SHADER` is a shader that is intended to run on the programmable geometry processor. A shader of type `?GL_FRAGMENT_SHADER` is a shader that is intended to run on the programmable fragment processor.

When created, a shader object's `?GL_SHADER_TYPE` parameter is set to either `?GL_VERTEX_SHADER` , `?GL_TESS_CONTROL_SHADER`, `?GL_TESS_EVALUATION_SHADER`, `?GL_GEOMETRY_SHADER` or `?GL_FRAGMENT_SHADER`, depending on the value of `ShaderType` .

See **external** documentation.

deleteProgram(Program) -> ok

Types:

Program = integer()

Deletes a program object

`gl:deleteProgram` frees the memory and invalidates the name associated with the program object specified by `Program.` This command effectively undoes the effects of a call to *gl:createProgram/0* .

If a program object is in use as part of current rendering state, it will be flagged for deletion, but it will not be deleted until it is no longer part of current state for any rendering context. If a program object to be deleted has shader objects

attached to it, those shader objects will be automatically detached but not deleted unless they have already been flagged for deletion by a previous call to *gl:deleteShader/1* . A value of 0 for `Program` will be silently ignored.

To determine whether a program object has been flagged for deletion, call *gl:getProgramiv/2* with arguments `Program` and `?GL_DELETE_STATUS`.

See **external** documentation.

deleteShader(Shader) -> ok

Types:

Shader = integer()

Deletes a shader object

`gl:deleteShader` frees the memory and invalidates the name associated with the shader object specified by `Shader` . This command effectively undoes the effects of a call to *gl:createShader/1* .

If a shader object to be deleted is attached to a program object, it will be flagged for deletion, but it will not be deleted until it is no longer attached to any program object, for any rendering context (i.e., it must be detached from wherever it was attached before it will be deleted). A value of 0 for `Shader` will be silently ignored.

To determine whether an object has been flagged for deletion, call *gl:getShaderiv/2* with arguments `Shader` and `?GL_DELETE_STATUS`.

See **external** documentation.

detachShader(Program, Shader) -> ok

Types:

Program = integer()

Shader = integer()

Detaches a shader object from a program object to which it is attached

`gl:detachShader` detaches the shader object specified by `Shader` from the program object specified by `Program` . This command can be used to undo the effect of the command *gl:attachShader/2* .

If `Shader` has already been flagged for deletion by a call to *gl:deleteShader/1* and it is not attached to any other program object, it will be deleted after it has been detached.

See **external** documentation.

disableVertexAttribArray(Index) -> ok

Types:

Index = integer()

Enable or disable a generic vertex attribute array

`gl:enableVertexAttribArray` enables the generic vertex attribute array specified by `Index` . `gl:disableVertexAttribArray` disables the generic vertex attribute array specified by `Index` . By default, all client-side capabilities are disabled, including all generic vertex attribute arrays. If enabled, the values in the generic vertex attribute array will be accessed and used for rendering when calls are made to vertex array commands such as *gl:drawArrays/3* , *gl:drawElements/4* , *gl:drawRangeElements/6* , see `glMultiDrawElements` , or *gl:multiDrawArrays/3* .

See **external** documentation.

enableVertexAttribArray(Index) -> ok

Types:

Index = integer()

See *disableVertexAttribArray/1*

getActiveAttrib(Program, Index, BufSize) -> {Size::integer(), Type::enum(), Name::string()}

Types:

Program = integer()

Index = integer()

BufSize = integer()

Returns information about an active attribute variable for the specified program object

`gl:getActiveAttrib` returns information about an active attribute variable in the program object specified by `Program` . The number of active attributes can be obtained by calling *gl:getProgramiv/2* with the value `?GL_ACTIVE_ATTRIBUTES`. A value of 0 for `Index` selects the first active attribute variable. Permissible values for `Index` range from 0 to the number of active attribute variables minus 1.

A vertex shader may use either built-in attribute variables, user-defined attribute variables, or both. Built-in attribute variables have a prefix of "gl_" and reference conventional OpenGL vertex attribtes (e.g., `Gl_Vertex` , `Gl_Normal` , etc., see the OpenGL Shading Language specification for a complete list.) User-defined attribute variables have arbitrary names and obtain their values through numbered generic vertex attributes. An attribute variable (either built-in or user-defined) is considered active if it is determined during the link operation that it may be accessed during program execution. Therefore, `Program` should have previously been the target of a call to *gl:linkProgram/1* , but it is not necessary for it to have been linked successfully.

The size of the character buffer required to store the longest attribute variable name in `Program` can be obtained by calling *gl:getProgramiv/2* with the value `?GL_ACTIVE_ATTRIBUTE_MAX_LENGTH` . This value should be used to allocate a buffer of sufficient size to store the returned attribute name. The size of this character buffer is passed in `BufSize` , and a pointer to this character buffer is passed in `Name` .

`gl:getActiveAttrib` returns the name of the attribute variable indicated by `Index` , storing it in the character buffer specified by `Name` . The string returned will be null terminated. The actual number of characters written into this buffer is returned in `Length` , and this count does not include the null termination character. If the length of the returned string is not required, a value of `?NULL` can be passed in the `Length` argument.

The `Type` argument specifies a pointer to a variable into which the attribute variable's data type will be written. The symbolic constants `?GL_FLOAT`, `?GL_FLOAT_VEC2`, `?GL_FLOAT_VEC3`, `?GL_FLOAT_VEC4`, `?GL_FLOAT_MAT2`, `?GL_FLOAT_MAT3`, `?GL_FLOAT_MAT4`, `?GL_FLOAT_MAT2x3`, `?GL_FLOAT_MAT2x4`, `?GL_FLOAT_MAT3x2` , `?GL_FLOAT_MAT3x4`, `?GL_FLOAT_MAT4x2`, `?GL_FLOAT_MAT4x3`, `?GL_INT` , `?GL_INT_VEC2`, `?GL_INT_VEC3`, `?GL_INT_VEC4`, `?GL_UNSIGNED_INT_VEC` , `?GL_UNSIGNED_INT_VEC2`, `?GL_UNSIGNED_INT_VEC3`, `?GL_UNSIGNED_INT_VEC4`, `?DOUBLE`, `?DOUBLE_VEC2`, `?DOUBLE_VEC3`, `?DOUBLE_VEC4`, `?DOUBLE_MAT2` , `?DOUBLE_MAT3`, `?DOUBLE_MAT4`, `?DOUBLE_MAT2x3`, `?DOUBLE_MAT2x4`, `?DOUBLE_MAT3x2`, `?DOUBLE_MAT3x4`, `?DOUBLE_MAT4x2`, or `?DOUBLE_MAT4x3` may be returned. The `Size` argument will return the size of the attribute, in units of the type returned in `Type` .

The list of active attribute variables may include both built-in attribute variables (which begin with the prefix "gl_") as well as user-defined attribute variable names.

This function will return as much information as it can about the specified active attribute variable. If no information is available, `Length` will be 0, and `Name` will be an empty string. This situation could occur if this function is called after a link operation that failed. If an error occurs, the return values `Length` , `Size` , `Type` , and `Name` will be unmodified.

See **external** documentation.

getActiveUniform(Program, Index, BufSize) -> {Size::integer(), Type::enum(), Name::string()}

Types:

Program = integer()

Index = integer()

BufSize = integer()

Returns information about an active uniform variable for the specified program object

`gl:getActiveUniform` returns information about an active uniform variable in the program object specified by `Program` . The number of active uniform variables can be obtained by calling *gl:getProgramiv/2* with the value `?GL_ACTIVE_UNIFORMS`. A value of 0 for `Index` selects the first active uniform variable. Permissible values for `Index` range from 0 to the number of active uniform variables minus 1.

Shaders may use either built-in uniform variables, user-defined uniform variables, or both. Built-in uniform variables have a prefix of "gl_" and reference existing OpenGL state or values derived from such state (e.g., `Gl_DepthRangeParameters` , see the OpenGL Shading Language specification for a complete list.) User-defined uniform variables have arbitrary names and obtain their values from the application through calls to *gl:uniform1f/2* . A uniform variable (either built-in or user-defined) is considered active if it is determined during the link operation that it may be accessed during program execution. Therefore, `Program` should have previously been the target of a call to *gl:linkProgram/1* , but it is not necessary for it to have been linked successfully.

The size of the character buffer required to store the longest uniform variable name in `Program` can be obtained by calling *gl:getProgramiv/2* with the value `?GL_ACTIVE_UNIFORM_MAX_LENGTH` . This value should be used to allocate a buffer of sufficient size to store the returned uniform variable name. The size of this character buffer is passed in `BufSize` , and a pointer to this character buffer is passed in `Name.`

`gl:getActiveUniform` returns the name of the uniform variable indicated by `Index` , storing it in the character buffer specified by `Name` . The string returned will be null terminated. The actual number of characters written into this buffer is returned in `Length` , and this count does not include the null termination character. If the length of the returned string is not required, a value of `?NULL` can be passed in the `Length` argument.

The `Type` argument will return a pointer to the uniform variable's data type. The symbolic constants returned for uniform types are shown in the table below.

```
Returned Symbolic ContantShader Uniform Type
?GL_FLOAT?float
?GL_FLOAT_VEC2 ?vec2
?GL_FLOAT_VEC3?vec3
?GL_FLOAT_VEC4?vec4
?GL_DOUBLE ?double
?GL_DOUBLE_VEC2?dvec2
?GL_DOUBLE_VEC3?dvec3
?GL_DOUBLE_VEC4 ?dvec4
?GL_INT?int
?GL_INT_VEC2?ivec2
?GL_INT_VEC3?ivec3
?GL_INT_VEC4?ivec4
?GL_UNSIGNED_INT ?unsigned int
?GL_UNSIGNED_INT_VEC2?uvec2
?GL_UNSIGNED_INT_VEC3?uvec3
?GL_UNSIGNED_INT_VEC4 ?uvec4
?GL_BOOL?bool
```

?GL_BOOL_VEC2?bvec2
?GL_BOOL_VEC3 ?bvec3
?GL_BOOL_VEC4?bvec4
?GL_FLOAT_MAT2?mat2
?GL_FLOAT_MAT3 ?mat3
?GL_FLOAT_MAT4?mat4
?GL_FLOAT_MAT2x3?mat2x3
?GL_FLOAT_MAT2x4 ?mat2x4
?GL_FLOAT_MAT3x2?mat3x2
?GL_FLOAT_MAT3x4?mat3x4
?GL_FLOAT_MAT4x2 ?mat4x2
?GL_FLOAT_MAT4x3?mat4x3
?GL_DOUBLE_MAT2?dmat2
?GL_DOUBLE_MAT3 ?dmat3
?GL_DOUBLE_MAT4?dmat4
?GL_DOUBLE_MAT2x3?dmat2x3
?GL_DOUBLE_MAT2x4 ?dmat2x4
?GL_DOUBLE_MAT3x2?dmat3x2
?GL_DOUBLE_MAT3x4?dmat3x4
?GL_DOUBLE_MAT4x2 ?dmat4x2
?GL_DOUBLE_MAT4x3?dmat4x3
?GL_SAMPLER_1D?sampler1D
?GL_SAMPLER_2D ?sampler2D
?GL_SAMPLER_3D?sampler3D
?GL_SAMPLER_CUBE?samplerCube
?GL_SAMPLER_1D_SHADOW ?sampler1DShadow
?GL_SAMPLER_2D_SHADOW?sampler2DShadow
?GL_SAMPLER_1D_ARRAY?sampler1DArray
?GL_SAMPLER_2D_ARRAY?sampler2DArray
?GL_SAMPLER_1D_ARRAY_SHADOW ?sampler1DArrayShadow
?GL_SAMPLER_2D_ARRAY_SHADOW ?sampler2DArrayShadow
?GL_SAMPLER_2D_MULTISAMPLE ?sampler2DMS
?GL_SAMPLER_2D_MULTISAMPLE_ARRAY?sampler2DMSArray
?GL_SAMPLER_CUBE_SHADOW?samplerCubeShadow
?GL_SAMPLER_BUFFER?samplerBuffer
?GL_SAMPLER_2D_RECT ?sampler2DRect
?GL_SAMPLER_2D_RECT_SHADOW ?sampler2DRectShadow
?GL_INT_SAMPLER_1D?isampler1D
?GL_INT_SAMPLER_2D?isampler2D
?GL_INT_SAMPLER_3D ?isampler3D
?GL_INT_SAMPLER_CUBE?isamplerCube
?GL_INT_SAMPLER_1D_ARRAY?isampler1DArray
?GL_INT_SAMPLER_2D_ARRAY?isampler2DArray
?GL_INT_SAMPLER_2D_MULTISAMPLE ?isampler2DMS
?GL_INT_SAMPLER_2D_MULTISAMPLE_ARRAY ?isampler2DMSArray
?GL_INT_SAMPLER_BUFFER?isamplerBuffer
?GL_INT_SAMPLER_2D_RECT?isampler2DRect
?GL_UNSIGNED_INT_SAMPLER_1D?usampler1D
?GL_UNSIGNED_INT_SAMPLER_2D ?usampler2D
?GL_UNSIGNED_INT_SAMPLER_3D?usampler3D
?GL_UNSIGNED_INT_SAMPLER_CUBE?usamplerCube
?GL_UNSIGNED_INT_SAMPLER_1D_ARRAY?usampler2DArray

?GL_UNSIGNED_INT_SAMPLER_2D_ARRAY?usampler2DArray
?GL_UNSIGNED_INT_SAMPLER_2D_MULTISAMPLE?usampler2DMS
?GL_UNSIGNED_INT_SAMPLER_2D_MULTISAMPLE_ARRAY?usampler2DMSArray
?GL_UNSIGNED_INT_SAMPLER_BUFFER?usamplerBuffer
?GL_UNSIGNED_INT_SAMPLER_2D_RECT?usampler2DRect
?GL_IMAGE_1D?image1D
?GL_IMAGE_2D ?image2D
?GL_IMAGE_3D?image3D
?GL_IMAGE_2D_RECT?image2DRect
?GL_IMAGE_CUBE ?imageCube
?GL_IMAGE_BUFFER?imageBuffer
?GL_IMAGE_1D_ARRAY?image1DArray
?GL_IMAGE_2D_ARRAY?image2DArray
?GL_IMAGE_2D_MULTISAMPLE ?image2DMS
?GL_IMAGE_2D_MULTISAMPLE_ARRAY ?image2DMSArray
?GL_INT_IMAGE_1D?iimage1D
?GL_INT_IMAGE_2D?iimage2D
?GL_INT_IMAGE_3D ?iimage3D
?GL_INT_IMAGE_2D_RECT?iimage2DRect
?GL_INT_IMAGE_CUBE?iimageCube
?GL_INT_IMAGE_BUFFER ?iimageBuffer
?GL_INT_IMAGE_1D_ARRAY?iimage1DArray
?GL_INT_IMAGE_2D_ARRAY?iimage2DArray
?GL_INT_IMAGE_2D_MULTISAMPLE?iimage2DMS
?GL_INT_IMAGE_2D_MULTISAMPLE_ARRAY ?iimage2DMSArray
?GL_UNSIGNED_INT_IMAGE_1D ?uimage1D
?GL_UNSIGNED_INT_IMAGE_2D?uimage2D
?GL_UNSIGNED_INT_IMAGE_3D?uimage3D
?GL_UNSIGNED_INT_IMAGE_2D_RECT?uimage2DRect
?GL_UNSIGNED_INT_IMAGE_CUBE ?uimageCube
?GL_UNSIGNED_INT_IMAGE_BUFFER ?uimageBuffer
?GL_UNSIGNED_INT_IMAGE_1D_ARRAY?uimage1DArray
?GL_UNSIGNED_INT_IMAGE_2D_ARRAY?uimage2DArray
?GL_UNSIGNED_INT_IMAGE_2D_MULTISAMPLE?uimage2DMS
?GL_UNSIGNED_INT_IMAGE_2D_MULTISAMPLE_ARRAY?uimage2DMSArray
?GL_UNSIGNED_INT_ATOMIC_COUNTER?atomic_uint

If one or more elements of an array are active, the name of the array is returned in `Name` , the type is returned in `Type` , and the `Size` parameter returns the highest array element index used, plus one, as determined by the compiler and/or linker. Only one active uniform variable will be reported for a uniform array.

Uniform variables that are declared as structures or arrays of structures will not be returned directly by this function. Instead, each of these uniform variables will be reduced to its fundamental components containing the "." and "[]" operators such that each of the names is valid as an argument to *gl:getUniformLocation/2* . Each of these reduced uniform variables is counted as one active uniform variable and is assigned an index. A valid name cannot be a structure, an array of structures, or a subcomponent of a vector or matrix.

The size of the uniform variable will be returned in `Size` . Uniform variables other than arrays will have a size of 1. Structures and arrays of structures will be reduced as described earlier, such that each of the names returned will be a data type in the earlier list. If this reduction results in an array, the size returned will be as described for uniform arrays; otherwise, the size returned will be 1.

The list of active uniform variables may include both built-in uniform variables (which begin with the prefix "gl_") as well as user-defined uniform variable names.

This function will return as much information as it can about the specified active uniform variable. If no information is available, `Length` will be 0, and `Name` will be an empty string. This situation could occur if this function is called after a link operation that failed. If an error occurs, the return values `Length` , `Size` , `Type` , and `Name` will be unmodified.

See **external** documentation.

`getAttachedShaders(Program, MaxCount) -> [integer()]`

Types:

`Program = integer()`
`MaxCount = integer()`

Returns the handles of the shader objects attached to a program object

`gl:getAttachedShaders` returns the names of the shader objects attached to `Program` . The names of shader objects that are attached to `Program` will be returned in `Shaders.` The actual number of shader names written into `Shaders` is returned in `Count.` If no shader objects are attached to `Program` , `Count` is set to 0. The maximum number of shader names that may be returned in `Shaders` is specified by `MaxCount` .

If the number of names actually returned is not required (for instance, if it has just been obtained by calling *gl:getProgramiv/2*), a value of `?NULL` may be passed for count. If no shader objects are attached to `Program` , a value of 0 will be returned in `Count` . The actual number of attached shaders can be obtained by calling *gl:getProgramiv/2* with the value `?GL_ATTACHED_SHADERS`.

See **external** documentation.

`getAttribLocation(Program, Name) -> integer()`

Types:

`Program = integer()`
`Name = string()`

Returns the location of an attribute variable

`gl:getAttribLocation` queries the previously linked program object specified by `Program` for the attribute variable specified by `Name` and returns the index of the generic vertex attribute that is bound to that attribute variable. If `Name` is a matrix attribute variable, the index of the first column of the matrix is returned. If the named attribute variable is not an active attribute in the specified program object or if `Name` starts with the reserved prefix "gl_", a value of -1 is returned.

The association between an attribute variable name and a generic attribute index can be specified at any time by calling *gl:bindAttribLocation/3* . Attribute bindings do not go into effect until *gl:linkProgram/1* is called. After a program object has been linked successfully, the index values for attribute variables remain fixed until the next link command occurs. The attribute values can only be queried after a link if the link was successful. `gl:getAttribLocation` returns the binding that actually went into effect the last time *gl:linkProgram/1* was called for the specified program object. Attribute bindings that have been specified since the last link operation are not returned by `gl:getAttribLocation`.

See **external** documentation.

`getProgramiv(Program, Pname) -> integer()`

Types:

`Program = integer()`
`Pname = enum()`

Returns a parameter from a program object

gl:getProgram returns in Params the value of a parameter for a specific program object. The following parameters are defined:

?GL_DELETE_STATUS: Params returns ?GL_TRUE if Program is currently flagged for deletion, and ?GL_FALSE otherwise.

?GL_LINK_STATUS: Params returns ?GL_TRUE if the last link operation on Program was successful, and ?GL_FALSE otherwise.

?GL_VALIDATE_STATUS: Params returns ?GL_TRUE or if the last validation operation on Program was successful, and ?GL_FALSE otherwise.

?GL_INFO_LOG_LENGTH: Params returns the number of characters in the information log for Program including the null termination character (i.e., the size of the character buffer required to store the information log). If Program has no information log, a value of 0 is returned.

?GL_ATTACHED_SHADERS: Params returns the number of shader objects attached to Program .

?GL_ACTIVE_ATOMIC_COUNTER_BUFFERS: Params returns the number of active attribute atomic counter buffers used by Program .

?GL_ACTIVE_ATTRIBUTES: Params returns the number of active attribute variables for Program .

?GL_ACTIVE_ATTRIBUTE_MAX_LENGTH: Params returns the length of the longest active attribute name for Program , including the null termination character (i.e., the size of the character buffer required to store the longest attribute name). If no active attributes exist, 0 is returned.

?GL_ACTIVE_UNIFORMS: Params returns the number of active uniform variables for Program .

?GL_ACTIVE_UNIFORM_MAX_LENGTH: Params returns the length of the longest active uniform variable name for Program , including the null termination character (i.e., the size of the character buffer required to store the longest uniform variable name). If no active uniform variables exist, 0 is returned.

?GL_PROGRAM_BINARY_LENGTH: Params returns the length of the program binary, in bytes that will be returned by a call to *gl:getProgramBinary/2* . When a progam's ?GL_LINK_STATUS is ?GL_FALSE, its program binary length is zero.

?GL_TRANSFORM_FEEDBACK_BUFFER_MODE: Params returns a symbolic constant indicating the buffer mode used when transform feedback is active. This may be ?GL_SEPARATE_ATTRIBS or ?GL_INTERLEAVED_ATTRIBS.

?GL_TRANSFORM_FEEDBACK_VARYINGS: Params returns the number of varying variables to capture in transform feedback mode for the program.

?GL_TRANSFORM_FEEDBACK_VARYING_MAX_LENGTH: Params returns the length of the longest variable name to be used for transform feedback, including the null-terminator.

?GL_GEOMETRY_VERTICES_OUT: Params returns the maximum number of vertices that the geometry shader in Program will output.

?GL_GEOMETRY_INPUT_TYPE: Params returns a symbolic constant indicating the primitive type accepted as input to the geometry shader contained in Program .

?GL_GEOMETRY_OUTPUT_TYPE: Params returns a symbolic constant indicating the primitive type that will be output by the geometry shader contained in Program .

See **external** documentation.

getProgramInfoLog(Program, BufSize) -> string()

Types:

Program = integer()

BufSize = integer()

Returns the information log for a program object

`gl:getProgramInfoLog` returns the information log for the specified program object. The information log for a program object is modified when the program object is linked or validated. The string that is returned will be null terminated.

`gl:getProgramInfoLog` returns in `InfoLog` as much of the information log as it can, up to a maximum of `MaxLength` characters. The number of characters actually returned, excluding the null termination character, is specified by `Length` . If the length of the returned string is not required, a value of `?NULL` can be passed in the `Length` argument. The size of the buffer required to store the returned information log can be obtained by calling *gl:getProgramiv/2* with the value `?GL_INFO_LOG_LENGTH` .

The information log for a program object is either an empty string, or a string containing information about the last link operation, or a string containing information about the last validation operation. It may contain diagnostic messages, warning messages, and other information. When a program object is created, its information log will be a string of length 0.

See **external** documentation.

`getShaderiv(Shader, Pname) -> integer()`

Types:

`Shader = integer()`

`Pname = enum()`

Returns a parameter from a shader object

`gl:getShader` returns in `Params` the value of a parameter for a specific shader object. The following parameters are defined:

`?GL_SHADER_TYPE`: `Params` returns `?GL_VERTEX_SHADER` if `Shader` is a vertex shader object, `?GL_GEOMETRY_SHADER` if `Shader` is a geometry shader object, and `?GL_FRAGMENT_SHADER` if `Shader` is a fragment shader object.

`?GL_DELETE_STATUS`: `Params` returns `?GL_TRUE` if `Shader` is currently flagged for deletion, and `?GL_FALSE` otherwise.

`?GL_COMPILE_STATUS`: `Params` returns `?GL_TRUE` if the last compile operation on `Shader` was successful, and `?GL_FALSE` otherwise.

`?GL_INFO_LOG_LENGTH`: `Params` returns the number of characters in the information log for `Shader` including the null termination character (i.e., the size of the character buffer required to store the information log). If `Shader` has no information log, a value of 0 is returned.

`?GL_SHADER_SOURCE_LENGTH`: `Params` returns the length of the concatenation of the source strings that make up the shader source for the `Shader` , including the null termination character. (i.e., the size of the character buffer required to store the shader source). If no source code exists, 0 is returned.

See **external** documentation.

`getShaderInfoLog(Shader, BufSize) -> string()`

Types:

`Shader = integer()`

`BufSize = integer()`

Returns the information log for a shader object

`gl:getShaderInfoLog` returns the information log for the specified shader object. The information log for a shader object is modified when the shader is compiled. The string that is returned will be null terminated.

gl:getShaderInfoLog returns in InfoLog as much of the information log as it can, up to a maximum of MaxLength characters. The number of characters actually returned, excluding the null termination character, is specified by Length . If the length of the returned string is not required, a value of ?NULL can be passed in the Length argument. The size of the buffer required to store the returned information log can be obtained by calling *gl:getShaderiv/2* with the value ?GL_INFO_LOG_LENGTH .

The information log for a shader object is a string that may contain diagnostic messages, warning messages, and other information about the last compile operation. When a shader object is created, its information log will be a string of length 0.

See **external** documentation.

getShaderSource(Shader, BufSize) -> string()

Types:

Shader = integer()

BufSize = integer()

Returns the source code string from a shader object

gl:getShaderSource returns the concatenation of the source code strings from the shader object specified by Shader . The source code strings for a shader object are the result of a previous call to *gl:shaderSource/2* . The string returned by the function will be null terminated.

gl:getShaderSource returns in Source as much of the source code string as it can, up to a maximum of BufSize characters. The number of characters actually returned, excluding the null termination character, is specified by Length . If the length of the returned string is not required, a value of ?NULL can be passed in the Length argument. The size of the buffer required to store the returned source code string can be obtained by calling *gl:getShaderiv/2* with the value ?GL_SHADER_SOURCE_LENGTH .

See **external** documentation.

getUniformLocation(Program, Name) -> integer()

Types:

Program = integer()

Name = string()

Returns the location of a uniform variable

gl:getUniformLocation returns an integer that represents the location of a specific uniform variable within a program object. Name must be a null terminated string that contains no white space. Name must be an active uniform variable name in Program that is not a structure, an array of structures, or a subcomponent of a vector or a matrix. This function returns -1 if Name does not correspond to an active uniform variable in Program , if Name starts with the reserved prefix "gl_", or if Name is associated with an atomic counter or a named uniform block.

Uniform variables that are structures or arrays of structures may be queried by calling gl:getUniformLocation for each field within the structure. The array element operator "[]" and the structure field operator "." may be used in Name in order to select elements within an array or fields within a structure. The result of using these operators is not allowed to be another structure, an array of structures, or a subcomponent of a vector or a matrix. Except if the last part of Name indicates a uniform variable array, the location of the first element of an array can be retrieved by using the name of the array, or by using the name appended by "[0]".

The actual locations assigned to uniform variables are not known until the program object is linked successfully. After linking has occurred, the command gl:getUniformLocation can be used to obtain the location of a uniform variable. This location value can then be passed to *gl:uniform1f/2* to set the value of the uniform variable or to *gl:getUniformfv/2* in order to query the current value of the uniform variable. After a program object has been linked

successfully, the index values for uniform variables remain fixed until the next link command occurs. Uniform variable locations and values can only be queried after a link if the link was successful.

See **external** documentation.

getUniformfv(Program, Location) -> matrix()

Types:

Program = integer()

Location = integer()

Returns the value of a uniform variable

`gl:getUniform` returns in `Params` the value(s) of the specified uniform variable. The type of the uniform variable specified by `Location` determines the number of values returned. If the uniform variable is defined in the shader as a boolean, int, or float, a single value will be returned. If it is defined as a vec2, ivec2, or bvec2, two values will be returned. If it is defined as a vec3, ivec3, or bvec3, three values will be returned, and so on. To query values stored in uniform variables declared as arrays, call `gl:getUniform` for each element of the array. To query values stored in uniform variables declared as structures, call `gl:getUniform` for each field in the structure. The values for uniform variables declared as a matrix will be returned in column major order.

The locations assigned to uniform variables are not known until the program object is linked. After linking has occurred, the command *gl:getUniformLocation/2* can be used to obtain the location of a uniform variable. This location value can then be passed to `gl:getUniform` in order to query the current value of the uniform variable. After a program object has been linked successfully, the index values for uniform variables remain fixed until the next link command occurs. The uniform variable values can only be queried after a link if the link was successful.

See **external** documentation.

getUniformiv(Program, Location) -> {integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer()}

Types:

Program = integer()

Location = integer()

See *getUniformfv/2*

getVertexAttribdv(Index, Pname) -> {float(), float(), float(), float()}

Types:

Index = integer()

Pname = enum()

Return a generic vertex attribute parameter

`gl:getVertexAttrib` returns in `Params` the value of a generic vertex attribute parameter. The generic vertex attribute to be queried is specified by `Index` , and the parameter to be queried is specified by `Pname` .

The accepted parameter names are as follows:

`?GL_VERTEX_ATTRIB_ARRAY_BUFFER_BINDING`: `Params` returns a single value, the name of the buffer object currently bound to the binding point corresponding to generic vertex attribute array `Index` . If no buffer object is bound, 0 is returned. The initial value is 0.

`?GL_VERTEX_ATTRIB_ARRAY_ENABLED`: `Params` returns a single value that is non-zero (true) if the vertex attribute array for `Index` is enabled and 0 (false) if it is disabled. The initial value is `?GL_FALSE`.

?GL_VERTEX_ATTRIB_ARRAY_SIZE: Params returns a single value, the size of the vertex attribute array for Index . The size is the number of values for each element of the vertex attribute array, and it will be 1, 2, 3, or 4. The initial value is 4.

?GL_VERTEX_ATTRIB_ARRAY_STRIDE: Params returns a single value, the array stride for (number of bytes between successive elements in) the vertex attribute array for Index . A value of 0 indicates that the array elements are stored sequentially in memory. The initial value is 0.

?GL_VERTEX_ATTRIB_ARRAY_TYPE: Params returns a single value, a symbolic constant indicating the array type for the vertex attribute array for Index . Possible values are ?GL_BYTE, ?GL_UNSIGNED_BYTE, ?GL_SHORT, ?GL_UNSIGNED_SHORT , ?GL_INT, ?GL_UNSIGNED_INT, ?GL_FLOAT, and ?GL_DOUBLE. The initial value is ?GL_FLOAT.

?GL_VERTEX_ATTRIB_ARRAY_NORMALIZED: Params returns a single value that is non-zero (true) if fixed-point data types for the vertex attribute array indicated by Index are normalized when they are converted to floating point, and 0 (false) otherwise. The initial value is ?GL_FALSE.

?GL_VERTEX_ATTRIB_ARRAY_INTEGER: Params returns a single value that is non-zero (true) if fixed-point data types for the vertex attribute array indicated by Index have integer data types, and 0 (false) otherwise. The initial value is 0 (?GL_FALSE).

?GL_VERTEX_ATTRIB_ARRAY_DIVISOR: Params returns a single value that is the frequency divisor used for instanced rendering. See *gl:vertexAttribDivisor/2* . The initial value is 0.

?GL_CURRENT_VERTEX_ATTRIB: Params returns four values that represent the current value for the generic vertex attribute specified by index. Generic vertex attribute 0 is unique in that it has no current state, so an error will be generated if Index is 0. The initial value for all other generic vertex attributes is (0,0,0,1).

gl:getVertexAttribdv and gl:getVertexAttribfv return the current attribute values as four single-precision floating-point values; gl:getVertexAttribiv reads them as floating-point values and converts them to four integer values; gl:getVertexAttribIiv and gl:getVertexAttribIuiv read and return them as signed or unsigned integer values, respectively; gl:getVertexAttribLdv reads and returns them as four double-precision floating-point values.

All of the parameters except ?GL_CURRENT_VERTEX_ATTRIB represent state stored in the currently bound vertex array object.

See **external** documentation.

getVertexAttribfv(Index, Pname) -> {float(), float(), float(), float()}

Types:

Index = integer()

Pname = enum()

See *getVertexAttribdv/2*

getVertexAttribiv(Index, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Index = integer()

Pname = enum()

See *getVertexAttribdv/2*

isProgram(Program) -> 0 | 1

Types:

Program = integer()

Determines if a name corresponds to a program object

`gl:isProgram` returns `?GL_TRUE` if `Program` is the name of a program object previously created with *gl:createProgram/0* and not yet deleted with *gl:deleteProgram/1* . If `Program` is zero or a non-zero value that is not the name of a program object, or if an error occurs, `gl:isProgram` returns `?GL_FALSE`.

See **external** documentation.

isShader(Shader) -> 0 | 1

Types:

Shader = integer()

Determines if a name corresponds to a shader object

`gl:isShader` returns `?GL_TRUE` if `Shader` is the name of a shader object previously created with *gl:createShader/1* and not yet deleted with *gl:deleteShader/1* . If `Shader` is zero or a non-zero value that is not the name of a shader object, or if an error occurs, `gl:isShader` returns `?GL_FALSE`.

See **external** documentation.

linkProgram(Program) -> ok

Types:

Program = integer()

Links a program object

`gl:linkProgram` links the program object specified by `Program` . If any shader objects of type `?GL_VERTEX_SHADER` are attached to `Program` , they will be used to create an executable that will run on the programmable vertex processor. If any shader objects of type `?GL_GEOMETRY_SHADER` are attached to `Program` , they will be used to create an executable that will run on the programmable geometry processor. If any shader objects of type `?GL_FRAGMENT_SHADER` are attached to `Program` , they will be used to create an executable that will run on the programmable fragment processor.

The status of the link operation will be stored as part of the program object's state. This value will be set to `?GL_TRUE` if the program object was linked without errors and is ready for use, and `?GL_FALSE` otherwise. It can be queried by calling *gl:getProgramiv/2* with arguments `Program` and `?GL_LINK_STATUS`.

As a result of a successful link operation, all active user-defined uniform variables belonging to `Program` will be initialized to 0, and each of the program object's active uniform variables will be assigned a location that can be queried by calling *gl:getUniformLocation/2* . Also, any active user-defined attribute variables that have not been bound to a generic vertex attribute index will be bound to one at this time.

Linking of a program object can fail for a number of reasons as specified in the `OpenGL Shading Language Specification` . The following lists some of the conditions that will cause a link error.

The number of active attribute variables supported by the implementation has been exceeded.

The storage limit for uniform variables has been exceeded.

The number of active uniform variables supported by the implementation has been exceeded.

The `main` function is missing for the vertex, geometry or fragment shader.

A varying variable actually used in the fragment shader is not declared in the same way (or is not declared at all) in the vertex shader, or geometry shader shader if present.

A reference to a function or variable name is unresolved.

A shared global is declared with two different types or two different initial values.

One or more of the attached shader objects has not been successfully compiled.

Binding a generic attribute matrix caused some rows of the matrix to fall outside the allowed maximum of ?GL_MAX_VERTEX_ATTRIBS.

Not enough contiguous vertex attribute slots could be found to bind attribute matrices.

The program object contains objects to form a fragment shader but does not contain objects to form a vertex shader.

The program object contains objects to form a geometry shader but does not contain objects to form a vertex shader.

The program object contains objects to form a geometry shader and the input primitive type, output primitive type, or maximum output vertex count is not specified in any compiled geometry shader object.

The program object contains objects to form a geometry shader and the input primitive type, output primitive type, or maximum output vertex count is specified differently in multiple geometry shader objects.

The number of active outputs in the fragment shader is greater than the value of ?GL_MAX_DRAW_BUFFERS .

The program has an active output assigned to a location greater than or equal to the value of ?GL_MAX_DUAL_SOURCE_DRAW_BUFFERS and has an active output assigned an index greater than or equal to one.

More than one varying out variable is bound to the same number and index.

The explicit binding assigments do not leave enough space for the linker to automatically assign a location for a varying out array, which requires multiple contiguous locations.

The `Count` specified by *gl:transformFeedbackVaryings/3* is non-zero, but the program object has no vertex or geometry shader.

Any variable name specified to *gl:transformFeedbackVaryings/3* in the `Varyings` array is not declared as an output in the vertex shader (or the geometry shader, if active).

Any two entries in the `Varyings` array given *gl:transformFeedbackVaryings/3* specify the same varying variable.

The total number of components to capture in any transform feedback varying variable is greater than the constant ?GL_MAX_TRANSFORM_FEEDBACK_SEPARATE_COMPONENTS and the buffer mode is ?SEPARATE_ATTRIBS.

When a program object has been successfully linked, the program object can be made part of current state by calling *gl:useProgram/1* . Whether or not the link operation was successful, the program object's information log will be overwritten. The information log can be retrieved by calling *gl:getProgramInfoLog/2* .

`gl:linkProgram` will also install the generated executables as part of the current rendering state if the link operation was successful and the specified program object is already currently in use as a result of a previous call to *gl:useProgram/1* . If the program object currently in use is relinked unsuccessfully, its link status will be set to ?GL_FALSE , but the executables and associated state will remain part of the current state until a subsequent call to `gl:useProgram` removes it from use. After it is removed from use, it cannot be made part of current state until it has been successfully relinked.

If `Program` contains shader objects of type ?GL_VERTEX_SHADER, and optionally of type ?GL_GEOMETRY_SHADER, but does not contain shader objects of type ?GL_FRAGMENT_SHADER , the vertex shader executable will be installed on the programmable vertex processor, the geometry shader executable, if present, will be installed on the programmable geometry processor, but no executable will be installed on the fragment processor. The results of rasterizing primitives with such a program will be undefined.

The program object's information log is updated and the program is generated at the time of the link operation. After the link operation, applications are free to modify attached shader objects, compile attached shader objects, detach shader objects, delete shader objects, and attach additional shader objects. None of these operations affects the information log or the program that is part of the program object.

See **external** documentation.

shaderSource(Shader, String) -> ok

Types:

Shader = integer()

String = [string()]

Replaces the source code in a shader object

`gl:shaderSource` sets the source code in `Shader` to the source code in the array of strings specified by `String` . Any source code previously stored in the shader object is completely replaced. The number of strings in the array is specified by `Count` . If `Length` is `?NULL`, each string is assumed to be null terminated. If `Length` is a value other than `?NULL`, it points to an array containing a string length for each of the corresponding elements of `String` . Each element in the `Length` array may contain the length of the corresponding string (the null character is not counted as part of the string length) or a value less than 0 to indicate that the string is null terminated. The source code strings are not scanned or parsed at this time; they are simply copied into the specified shader object.

See **external** documentation.

useProgram(Program) -> ok

Types:

Program = integer()

Installs a program object as part of current rendering state

`gl:useProgram` installs the program object specified by `Program` as part of current rendering state. One or more executables are created in a program object by successfully attaching shader objects to it with *gl:attachShader/2* , successfully compiling the shader objects with *gl:compileShader/1* , and successfully linking the program object with *gl:linkProgram/1* .

A program object will contain an executable that will run on the vertex processor if it contains one or more shader objects of type `?GL_VERTEX_SHADER` that have been successfully compiled and linked. A program object will contain an executable that will run on the geometry processor if it contains one or more shader objects of type `?GL_GEOMETRY_SHADER` that have been successfully compiled and linked. Similarly, a program object will contain an executable that will run on the fragment processor if it contains one or more shader objects of type `?GL_FRAGMENT_SHADER` that have been successfully compiled and linked.

While a program object is in use, applications are free to modify attached shader objects, compile attached shader objects, attach additional shader objects, and detach or delete shader objects. None of these operations will affect the executables that are part of the current state. However, relinking the program object that is currently in use will install the program object as part of the current rendering state if the link operation was successful (see *gl:linkProgram/1*). If the program object currently in use is relinked unsuccessfully, its link status will be set to `?GL_FALSE`, but the executables and associated state will remain part of the current state until a subsequent call to `gl:useProgram` removes it from use. After it is removed from use, it cannot be made part of current state until it has been successfully relinked.

If `Program` is zero, then the current rendering state refers to an `invalid` program object and the results of shader execution are undefined. However, this is not an error.

If `Program` does not contain shader objects of type `?GL_FRAGMENT_SHADER`, an executable will be installed on the vertex, and possibly geometry processors, but the results of fragment shader execution will be undefined.

See **external** documentation.

uniform1f(Location, V0) -> ok

Types:

Location = integer()

V0 = float()

Specify the value of a uniform variable for the current program object

`gl:uniform` modifies the value of a uniform variable or a uniform variable array. The location of the uniform variable to be modified is specified by `Location` , which should be a value returned by *gl:getUniformLocation/2* . `gl:uniform` operates on the program object that was made part of current state by calling *gl:useProgram/1* .

The commands `gl:uniform{1|2|3|4}{f|i|ui}` are used to change the value of the uniform variable specified by `Location` using the values passed as arguments. The number specified in the command should match the number of components in the data type of the specified uniform variable (e.g., `1` for float, int, unsigned int, bool; `2` for vec2, ivec2, uvec2, bvec2, etc.). The suffix `f` indicates that floating-point values are being passed; the suffix `i` indicates that integer values are being passed; the suffix `ui` indicates that unsigned integer values are being passed, and this type should also match the data type of the specified uniform variable. The `i` variants of this function should be used to provide values for uniform variables defined as int, ivec2 , ivec3, ivec4, or arrays of these. The `ui` variants of this function should be used to provide values for uniform variables defined as unsigned int, uvec2, uvec3, uvec4, or arrays of these. The `f` variants should be used to provide values for uniform variables of type float, vec2, vec3, vec4, or arrays of these. Either the `i`, `ui` or `f` variants may be used to provide values for uniform variables of type bool, bvec2 , bvec3, bvec4, or arrays of these. The uniform variable will be set to false if the input value is 0 or 0.0f, and it will be set to true otherwise.

All active uniform variables defined in a program object are initialized to 0 when the program object is linked successfully. They retain the values assigned to them by a call to `gl:uniform` until the next successful link operation occurs on the program object, when they are once again initialized to 0.

The commands `gl:uniform{1|2|3|4}{f|i|ui}v` can be used to modify a single uniform variable or a uniform variable array. These commands pass a count and a pointer to the values to be loaded into a uniform variable or a uniform variable array. A count of 1 should be used if modifying the value of a single uniform variable, and a count of 1 or greater can be used to modify an entire array or part of an array. When loading `n` elements starting at an arbitrary position `m` in a uniform variable array, elements `m` + `n` - 1 in the array will be replaced with the new values. If `M` + `N` - 1 is larger than the size of the uniform variable array, values for all array elements beyond the end of the array will be ignored. The number specified in the name of the command indicates the number of components for each element in `Value` , and it should match the number of components in the data type of the specified uniform variable (e.g., `1` for float, int, bool; `2` for vec2, ivec2, bvec2, etc.). The data type specified in the name of the command must match the data type for the specified uniform variable as described previously for `gl:uniform{1|2|3|4}{f|i|ui}`.

For uniform variable arrays, each element of the array is considered to be of the type indicated in the name of the command (e.g., `gl:uniform3f` or `gl:uniform3fv` can be used to load a uniform variable array of type vec3). The number of elements of the uniform variable array to be modified is specified by `Count`

The commands `gl:uniformMatrix{2|3|4|2x3|3x2|2x4|4x2|3x4|4x3}fv` are used to modify a matrix or an array of matrices. The numbers in the command name are interpreted as the dimensionality of the matrix. The number `2` indicates a 2 × 2 matrix (i.e., 4 values), the number `3` indicates a 3 × 3 matrix (i.e., 9 values), and the number `4` indicates a 4 × 4 matrix (i.e., 16 values). Non-square matrix dimensionality is explicit, with the first number representing the number of columns and the second number representing the number of rows. For example, `2x4` indicates a 2 × 4 matrix with 2 columns and 4 rows (i.e., 8 values). If `Transpose` is `?GL_FALSE`, each matrix is assumed to be supplied in column major order. If `Transpose` is `?GL_TRUE`, each matrix is assumed to be supplied in row major order. The `Count` argument indicates the number of matrices to be passed. A count of 1 should be used if modifying the value of a single matrix, and a count greater than 1 can be used to modify an array of matrices.

See **external** documentation.

`uniform2f(Location, V0, V1) -> ok`

Types:

```
Location = integer()
V0 = float()
V1 = float()
```

See *uniform1f/2*

uniform3f(Location, V0, V1, V2) -> ok

Types:

Location = integer()
V0 = float()
V1 = float()
V2 = float()

See *uniform1f/2*

uniform4f(Location, V0, V1, V2, V3) -> ok

Types:

Location = integer()
V0 = float()
V1 = float()
V2 = float()
V3 = float()

See *uniform1f/2*

uniform1i(Location, V0) -> ok

Types:

Location = integer()
V0 = integer()

See *uniform1f/2*

uniform2i(Location, V0, V1) -> ok

Types:

Location = integer()
V0 = integer()
V1 = integer()

See *uniform1f/2*

uniform3i(Location, V0, V1, V2) -> ok

Types:

Location = integer()
V0 = integer()
V1 = integer()
V2 = integer()

See *uniform1f/2*

uniform4i(Location, V0, V1, V2, V3) -> ok

Types:

Location = integer()

```
V0 = integer()
V1 = integer()
V2 = integer()
V3 = integer()
```

See *uniform1f/2*

uniform1fv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [float()]
```

See *uniform1f/2*

uniform2fv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{float(), float()}]
```

See *uniform1f/2*

uniform3fv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{float(), float(), float()}]
```

See *uniform1f/2*

uniform4fv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{float(), float(), float(), float()}]
```

See *uniform1f/2*

uniform1iv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [integer()]
```

See *uniform1f/2*

uniform2iv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{integer(), integer()}]
```

See *uniform1f/2*

uniform3iv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{integer(), integer(), integer()}]
```

See *uniform1f/2*

uniform4iv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{integer(), integer(), integer(), integer()}]
```

See *uniform1f/2*

uniformMatrix2fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix3fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float()}]
```

See *uniform1f/2*

uniformMatrix4fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *uniform1f/2*

validateProgram(Program) -> ok

Types:

```
Program = integer()
```

Validates a program object

`gl:validateProgram` checks to see whether the executables contained in `Program` can execute given the current OpenGL state. The information generated by the validation process will be stored in `Program` 's information log. The validation information may consist of an empty string, or it may be a string containing information about how the

current program object interacts with the rest of current OpenGL state. This provides a way for OpenGL implementers to convey more information about why the current program is inefficient, suboptimal, failing to execute, and so on.

The status of the validation operation will be stored as part of the program object's state. This value will be set to ?GL_TRUE if the validation succeeded, and ?GL_FALSE otherwise. It can be queried by calling *gl:getProgramiv/2* with arguments Program and ?GL_VALIDATE_STATUS. If validation is successful, Program is guaranteed to execute given the current state. Otherwise, Program is guaranteed to not execute.

This function is typically useful only during application development. The informational string stored in the information log is completely implementation dependent; therefore, an application should not expect different OpenGL implementations to produce identical information strings.

See **external** documentation.

vertexAttrib1d(Index, X) -> ok

Types:

```
Index = integer()
X = float()
```

Specifies the value of a generic vertex attribute

The gl:vertexAttrib family of entry points allows an application to pass generic vertex attributes in numbered locations.

Generic attributes are defined as four-component values that are organized into an array. The first entry of this array is numbered 0, and the size of the array is specified by the implementation-dependent constant ?GL_MAX_VERTEX_ATTRIBS. Individual elements of this array can be modified with a gl:vertexAttrib call that specifies the index of the element to be modified and a value for that element.

These commands can be used to specify one, two, three, or all four components of the generic vertex attribute specified by Index . A 1 in the name of the command indicates that only one value is passed, and it will be used to modify the first component of the generic vertex attribute. The second and third components will be set to 0, and the fourth component will be set to 1. Similarly, a 2 in the name of the command indicates that values are provided for the first two components, the third component will be set to 0, and the fourth component will be set to 1. A 3 in the name of the command indicates that values are provided for the first three components and the fourth component will be set to 1, whereas a 4 in the name indicates that values are provided for all four components.

The letters s, f, i, d, ub, us, and ui indicate whether the arguments are of type short, float, int, double, unsigned byte, unsigned short, or unsigned int. When v is appended to the name, the commands can take a pointer to an array of such values.

Additional capitalized letters can indicate further alterations to the default behavior of the glVertexAttrib function:

The commands containing N indicate that the arguments will be passed as fixed-point values that are scaled to a normalized range according to the component conversion rules defined by the OpenGL specification. Signed values are understood to represent fixed-point values in the range [-1,1], and unsigned values are understood to represent fixed-point values in the range [0,1].

The commands containing I indicate that the arguments are extended to full signed or unsigned integers.

The commands containing P indicate that the arguments are stored as packed components within a larger natural type.

The commands containing L indicate that the arguments are full 64-bit quantities and should be passed directly to shader inputs declared as 64-bit double precision types.

OpenGL Shading Language attribute variables are allowed to be of type mat2, mat3, or mat4. Attributes of these types may be loaded using the gl:vertexAttrib entry points. Matrices must be loaded into successive generic attribute slots in column major order, with one column of the matrix in each generic attribute slot.

A user-defined attribute variable declared in a vertex shader can be bound to a generic attribute index by calling *gl:bindAttribLocation/3* . This allows an application to use more descriptive variable names in a vertex shader. A subsequent change to the specified generic vertex attribute will be immediately reflected as a change to the corresponding attribute variable in the vertex shader.

The binding between a generic vertex attribute index and a user-defined attribute variable in a vertex shader is part of the state of a program object, but the current value of the generic vertex attribute is not. The value of each generic vertex attribute is part of current state, just like standard vertex attributes, and it is maintained even if a different program object is used.

An application may freely modify generic vertex attributes that are not bound to a named vertex shader attribute variable. These values are simply maintained as part of current state and will not be accessed by the vertex shader. If a generic vertex attribute bound to an attribute variable in a vertex shader is not updated while the vertex shader is executing, the vertex shader will repeatedly use the current value for the generic vertex attribute.

See **external** documentation.

`vertexAttrib1dv(Index::integer(), V) -> ok`

Types:

`V = {X::float()}`

Equivalent to *vertexAttrib1d(Index, X)*.

`vertexAttrib1f(Index, X) -> ok`

Types:

`Index = integer()`

`X = float()`

See *vertexAttrib1d/2*

`vertexAttrib1fv(Index::integer(), V) -> ok`

Types:

`V = {X::float()}`

Equivalent to *vertexAttrib1f(Index, X)*.

`vertexAttrib1s(Index, X) -> ok`

Types:

`Index = integer()`

`X = integer()`

See *vertexAttrib1d/2*

`vertexAttrib1sv(Index::integer(), V) -> ok`

Types:

`V = {X::integer()}`

Equivalent to *vertexAttrib1s(Index, X)*.

`vertexAttrib2d(Index, X, Y) -> ok`

Types:

`Index = integer()`

```
X = float()
Y = float()
```

See *vertexAttrib1d/2*

vertexAttrib2dv(Index::integer(), V) -> ok

Types:

```
V = {X::float(), Y::float()}
```

Equivalent to *vertexAttrib2d(Index, X, Y)*.

vertexAttrib2f(Index, X, Y) -> ok

Types:

```
Index = integer()
X = float()
Y = float()
```

See *vertexAttrib1d/2*

vertexAttrib2fv(Index::integer(), V) -> ok

Types:

```
V = {X::float(), Y::float()}
```

Equivalent to *vertexAttrib2f(Index, X, Y)*.

vertexAttrib2s(Index, X, Y) -> ok

Types:

```
Index = integer()
X = integer()
Y = integer()
```

See *vertexAttrib1d/2*

vertexAttrib2sv(Index::integer(), V) -> ok

Types:

```
V = {X::integer(), Y::integer()}
```

Equivalent to *vertexAttrib2s(Index, X, Y)*.

vertexAttrib3d(Index, X, Y, Z) -> ok

Types:

```
Index = integer()
X = float()
Y = float()
Z = float()
```

See *vertexAttrib1d/2*

vertexAttrib3dv(Index::integer(), V) -> ok

Types:

```
V = {X::float(), Y::float(), Z::float()}
```

Equivalent to *vertexAttrib3d(Index, X, Y, Z)*.

vertexAttrib3f(Index, X, Y, Z) -> ok

Types:

```
Index = integer()
X = float()
Y = float()
Z = float()
```

See *vertexAttrib1d/2*

vertexAttrib3fv(Index::integer(), V) -> ok

Types:

```
V = {X::float(), Y::float(), Z::float()}
```

Equivalent to *vertexAttrib3f(Index, X, Y, Z)*.

vertexAttrib3s(Index, X, Y, Z) -> ok

Types:

```
Index = integer()
X = integer()
Y = integer()
Z = integer()
```

See *vertexAttrib1d/2*

vertexAttrib3sv(Index::integer(), V) -> ok

Types:

```
V = {X::integer(), Y::integer(), Z::integer()}
```

Equivalent to *vertexAttrib3s(Index, X, Y, Z)*.

vertexAttrib4Nbv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttrib4Niv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttrib4Nsv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttrib4Nub(Index, X, Y, Z, W) -> ok

Types:

```
Index = integer()
X = integer()
Y = integer()
Z = integer()
W = integer()
```

See *vertexAttrib1d/2*

vertexAttrib4Nubv(Index::integer(), V) -> ok

Types:

```
V = {X::integer(), Y::integer(), Z::integer(), W::integer()}
```

Equivalent to *vertexAttrib4Nub(Index, X, Y, Z, W)*.

vertexAttrib4Nuiv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttrib4Nusv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttrib4bv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttrib4d(Index, X, Y, Z, W) -> ok

Types:

```
Index = integer()
X = float()
Y = float()
Z = float()
W = float()
```

See *vertexAttrib1d/2*

vertexAttrib4dv(Index::integer(), V) -> ok

Types:

V = {X::float(), Y::float(), Z::float(), W::float()}

Equivalent to *vertexAttrib4d(Index, X, Y, Z, W)*.

vertexAttrib4f(Index, X, Y, Z, W) -> ok

Types:

Index = integer()
X = float()
Y = float()
Z = float()
W = float()

See *vertexAttrib1d/2*

vertexAttrib4fv(Index::integer(), V) -> ok

Types:

V = {X::float(), Y::float(), Z::float(), W::float()}

Equivalent to *vertexAttrib4f(Index, X, Y, Z, W)*.

vertexAttrib4iv(Index, V) -> ok

Types:

Index = integer()
V = {integer(), integer(), integer(), integer()}

See *vertexAttrib1d/2*

vertexAttrib4s(Index, X, Y, Z, W) -> ok

Types:

Index = integer()
X = integer()
Y = integer()
Z = integer()
W = integer()

See *vertexAttrib1d/2*

vertexAttrib4sv(Index::integer(), V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer(), W::integer()}

Equivalent to *vertexAttrib4s(Index, X, Y, Z, W)*.

vertexAttrib4ubv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttrib4uiv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttrib4usv(Index, V) -> ok

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

See *vertexAttrib1d/2*

vertexAttribPointer(Index, Size, Type, Normalized, Stride, Pointer) -> ok

Types:

```
Index = integer()
Size = integer()
Type = enum()
Normalized = 0 | 1
Stride = integer()
Pointer = offset() | mem()
```

Define an array of generic vertex attribute data

`gl:vertexAttribPointer`, `gl:vertexAttribIPointer` and `gl:vertexAttribLPointer` specify the location and data format of the array of generic vertex attributes at index `Index` to use when rendering. `Size` specifies the number of components per attribute and must be 1, 2, 3, 4, or `?GL_BGRA`. `Type` specifies the data type of each component, and `Stride` specifies the byte stride from one attribute to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays.

For `gl:vertexAttribPointer`, if `Normalized` is set to `?GL_TRUE`, it indicates that values stored in an integer format are to be mapped to the range [-1,1] (for signed values) or [0,1] (for unsigned values) when they are accessed and converted to floating point. Otherwise, values will be converted to floats directly without normalization.

For `gl:vertexAttribIPointer`, only the integer types `?GL_BYTE`, `?GL_UNSIGNED_BYTE` , `?GL_SHORT`, `?GL_UNSIGNED_SHORT`, `?GL_INT`, `?GL_UNSIGNED_INT` are accepted. Values are always left as integer values.

`gl:vertexAttribLPointer` specifies state for a generic vertex attribute array associated with a shader attribute variable declared with 64-bit double precision components. `Type` must be `?GL_DOUBLE`. `Index` , `Size` , and `Stride` behave as described for `gl:vertexAttribPointer` and `gl:vertexAttribIPointer`.

If `Pointer` is not NULL, a non-zero named buffer object must be bound to the `?GL_ARRAY_BUFFER` target (see *gl:bindBuffer/2*), otherwise an error is generated. `Pointer` is treated as a byte offset into the buffer object's data store. The buffer object binding (`?GL_ARRAY_BUFFER_BINDING`) is saved as generic vertex attribute array state (`?GL_VERTEX_ATTRIB_ARRAY_BUFFER_BINDING`) for index `Index` .

When a generic vertex attribute array is specified, `Size` , `Type` , `Normalized` , `Stride` , and `Pointer` are saved as vertex array state, in addition to the current vertex array buffer object binding.

To enable and disable a generic vertex attribute array, call *gl:disableVertexAttribArray/1* and *gl:disableVertexAttribArray/1* with `Index` . If enabled, the generic vertex attribute array is used when *gl:drawArrays/3* , *gl:multiDrawArrays/3* , *gl:drawElements/4* , see `glMultiDrawElements`, or *gl:drawRangeElements/6* is called.

See **external** documentation.

uniformMatrix2x3fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix3x2fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix2x4fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *uniform1f/2*

uniformMatrix4x2fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *uniform1f/2*

uniformMatrix3x4fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix4x3fv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float()}]
```

See *uniform1f/2*

colorMaski(Index, R, G, B, A) -> ok

Types:

```
Index = integer()
R = 0 | 1
G = 0 | 1
B = 0 | 1
A = 0 | 1
```

glColorMaski

See **external** documentation.

getBooleani_v(Target, Index) -> [0 | 1]

Types:

```
Target = enum()
Index = integer()
```

See *getBooleanv/1*

getIntegeri_v(Target, Index) -> [integer()]

Types:

```
Target = enum()
Index = integer()
```

See *getBooleanv/1*

enablei(Target, Index) -> ok

Types:

```
Target = enum()
Index = integer()
```

See *enable/1*

disablei(Target, Index) -> ok

Types:

```
Target = enum()
Index = integer()
```

glEnablei

See **external** documentation.

isEnabledi(Target, Index) -> 0 | 1

Types:

Target = enum()

Index = integer()

glIsEnabledi

See **external** documentation.

beginTransformFeedback(PrimitiveMode) -> ok

Types:

PrimitiveMode = enum()

Start transform feedback operation

Transform feedback mode captures the values of varying variables written by the vertex shader (or, if active, the geometry shader). Transform feedback is said to be active after a call to `gl:beginTransformFeedback` until a subsequent call to *gl:beginTransformFeedback/1* . Transform feedback commands must be paired.

If no geometry shader is present, while transform feedback is active the `Mode` parameter to *gl:drawArrays/3* must match those specified in the following table:

`Transform Feedback` `PrimitiveMode`	`Allowed Render Primitive` `Modes`
`?GL_POINTS`	`?GL_POINTS`
`?GL_LINES`	`?GL_LINES`, `?GL_LINE_LOOP`, `?GL_LINE_STRIP` , `?GL_LINES_ADJACENCY`, `?GL_LINE_STRIP_ADJACENCY`
`?GL_TRIANGLES`	`?GL_TRIANGLES`, `?GL_TRIANGLE_STRIP`, `?GL_TRIANGLE_FAN`, `?GL_TRIANGLES_ADJACENCY` , `?GL_TRIANGLE_STRIP_ADJACENCY`

If a geometry shader is present, the output primitive type from the geometry shader must match those provided in the following table:

`Transform Feedback` `PrimitiveMode`	`Allowed Geometry Shader Output Primitive Type`
`?GL_POINTS`	`?points`
`?GL_LINES`	`?line_strip`
`?GL_TRIANGLES`	`?triangle_strip`

See **external** documentation.

endTransformFeedback() -> ok

See *beginTransformFeedback/1*

bindBufferRange(Target, Index, Buffer, Offset, Size) -> ok

Types:

Target = enum()

Index = integer()

Buffer = integer()

Offset = integer()

Size = integer()

Bind a range within a buffer object to an indexed buffer target

`gl:bindBufferRange` binds a range the buffer object `Buffer` represented by `Offset` and `Size` to the binding point at index `Index` of the array of targets specified by `Target` . Each `Target` represents an indexed array of buffer binding points, as well as a single general binding point that can be used by other buffer manipulation functions

such as *gl:bindBuffer/2* or see `glMapBuffer`. In addition to binding a range of `Buffer` to the indexed buffer binding target, `gl:bindBufferBase` also binds the range to the generic buffer binding point specified by `Target` .

`Offset` specifies the offset in basic machine units into the buffer object `Buffer` and `Size` specifies the amount of data that can be read from the buffer object while used as an indexed target.

See **external** documentation.

`bindBufferBase(Target, Index, Buffer) -> ok`

Types:

```
Target = enum()
Index = integer()
Buffer = integer()
```

Bind a buffer object to an indexed buffer target

`gl:bindBufferBase` binds the buffer object `Buffer` to the binding point at index `Index` of the array of targets specified by `Target` . Each `Target` represents an indexed array of buffer binding points, as well as a single general binding point that can be used by other buffer manipulation functions such as *gl:bindBuffer/2* or see `glMapBuffer`. In addition to binding `Buffer` to the indexed buffer binding target, `gl:bindBufferBase` also binds `Buffer` to the generic buffer binding point specified by `Target` .

See **external** documentation.

`transformFeedbackVaryings(Program, Varyings, BufferMode) -> ok`

Types:

```
Program = integer()
Varyings = [string()]
BufferMode = enum()
```

Specify values to record in transform feedback buffers

The names of the vertex or geometry shader outputs to be recorded in transform feedback mode are specified using `gl:transformFeedbackVaryings`. When a geometry shader is active, transform feedback records the values of selected geometry shader output variables from the emitted vertices. Otherwise, the values of the selected vertex shader outputs are recorded.

The state set by `gl:tranformFeedbackVaryings` is stored and takes effect next time *gl:linkProgram/1* is called on `Program` . When *gl:linkProgram/1* is called, `Program` is linked so that the values of the specified varying variables for the vertices of each primitive generated by the GL are written to a single buffer object if `BufferMode` is `?GL_INTERLEAVED_ATTRIBS` or multiple buffer objects if `BufferMode` is `?GL_SEPARATE_ATTRIBS` .

In addition to the errors generated by `gl:transformFeedbackVaryings`, the program `Program` will fail to link if:

The count specified by `gl:transformFeedbackVaryings` is non-zero, but the program object has no vertex or geometry shader.

Any variable name specified in the `Varyings` array is not declared as an output in the vertex shader (or the geometry shader, if active).

Any two entries in the `Varyings` array specify the same varying variable.

The total number of components to capture in any varying variable in `Varyings` is greater than the constant `?GL_MAX_TRANSFORM_FEEDBACK_SEPARATE_COMPONENTS` and the buffer mode is `?GL_SEPARATE_ATTRIBS`.

The total number of components to capture is greater than the constant ? GL_MAX_TRANSFORM_FEEDBACK_INTERLEAVED_COMPONENTS and the buffer mode is ? GL_INTERLEAVED_ATTRIBS.

See **external** documentation.

getTransformFeedbackVarying(Program, Index, BufSize) -> {Size::integer(), Type::enum(), Name::string()}

Types:

Program = integer()
Index = integer()
BufSize = integer()

Retrieve information about varying variables selected for transform feedback

Information about the set of varying variables in a linked program that will be captured during transform feedback may be retrieved by calling gl:getTransformFeedbackVarying. gl:getTransformFeedbackVarying provides information about the varying variable selected by Index . An Index of 0 selects the first varying variable specified in the Varyings array passed to *gl:transformFeedbackVaryings/3* , and an Index of ? GL_TRANSFORM_FEEDBACK_VARYINGS-1 selects the last such variable.

The name of the selected varying is returned as a null-terminated string in Name . The actual number of characters written into Name , excluding the null terminator, is returned in Length . If Length is NULL, no length is returned. The maximum number of characters that may be written into Name , including the null terminator, is specified by BufSize .

The length of the longest varying name in program is given by ? GL_TRANSFORM_FEEDBACK_VARYING_MAX_LENGTH , which can be queried with *gl:getProgramiv/2* .

For the selected varying variable, its type is returned into Type . The size of the varying is returned into Size . The value in Size is in units of the type returned in Type . The type returned can be any of the scalar, vector, or matrix attribute types returned by *gl:getActiveAttrib/3* . If an error occurred, the return parameters Length , Size , Type and Name will be unmodified. This command will return as much information about the varying variables as possible. If no information is available, Length will be set to zero and Name will be an empty string. This situation could arise if gl:getTransformFeedbackVarying is called after a failed link.

See **external** documentation.

clampColor(Target, Clamp) -> ok

Types:

Target = enum()
Clamp = enum()

specify whether data read via

gl:readPixels/7 should be clamped

gl:clampColor controls color clamping that is performed during *gl:readPixels/7* . Target must be ? GL_CLAMP_READ_COLOR. If Clamp is ?GL_TRUE, read color clamping is enabled; if Clamp is ?GL_FALSE, read color clamping is disabled. If Clamp is ?GL_FIXED_ONLY, read color clamping is enabled only if the selected read buffer has fixed point components and disabled otherwise.

See **external** documentation.

beginConditionalRender(Id, Mode) -> ok

Types:

```
Id = integer()
Mode = enum()
```

Start conditional rendering

Conditional rendering is started using `gl:beginConditionalRender` and ended using `gl:endConditionalRender` . During conditional rendering, all vertex array commands, as well as *gl:clear/1* and *gl:clearBufferiv/3* have no effect if the (`?GL_SAMPLES_PASSED`) result of the query object `Id` is zero, or if the (`?GL_ANY_SAMPLES_PASSED`) result is `?GL_FALSE` . The results of commands setting the current vertex state, such as *gl:vertexAttrib1d/2* are undefined. If the (`?GL_SAMPLES_PASSED`) result is non-zero or if the (`?GL_ANY_SAMPLES_PASSED`) result is `?GL_TRUE`, such commands are not discarded. The `Id` parameter to `gl:beginConditionalRender` must be the name of a query object previously returned from a call to *gl:genQueries/1* . `Mode` specifies how the results of the query object are to be interpreted. If `Mode` is `?GL_QUERY_WAIT`, the GL waits for the results of the query to be available and then uses the results to determine if subsequent rendering commands are discarded. If `Mode` is `?GL_QUERY_NO_WAIT`, the GL may choose to unconditionally execute the subsequent rendering commands without waiting for the query to complete.

If `Mode` is `?GL_QUERY_BY_REGION_WAIT`, the GL will also wait for occlusion query results and discard rendering commands if the result of the occlusion query is zero. If the query result is non-zero, subsequent rendering commands are executed, but the GL may discard the results of the commands for any region of the framebuffer that did not contribute to the sample count in the specified occlusion query. Any such discarding is done in an implementation-dependent manner, but the rendering command results may not be discarded for any samples that contributed to the occlusion query sample count. If `Mode` is `?GL_QUERY_BY_REGION_NO_WAIT`, the GL operates as in `?GL_QUERY_BY_REGION_WAIT` , but may choose to unconditionally execute the subsequent rendering commands without waiting for the query to complete.

See **external** documentation.

endConditionalRender() -> ok

See *beginConditionalRender/2*

vertexAttribIPointer(Index, Size, Type, Stride, Pointer) -> ok

Types:

```
Index = integer()
Size = integer()
Type = enum()
Stride = integer()
Pointer = offset() | mem()
```

glVertexAttribIPointer

See **external** documentation.

getVertexAttribIiv(Index, Pname) -> {integer(), integer(), integer(), integer()}

Types:

```
Index = integer()
Pname = enum()
```

See *getVertexAttribdv/2*

getVertexAttribIuiv(Index, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Index = integer()

Pname = enum()

glGetVertexAttribI

See **external** documentation.

vertexAttribI1i(Index, X) -> ok

Types:

Index = integer()

X = integer()

See *vertexAttrib1d/2*

vertexAttribI2i(Index, X, Y) -> ok

Types:

Index = integer()

X = integer()

Y = integer()

See *vertexAttrib1d/2*

vertexAttribI3i(Index, X, Y, Z) -> ok

Types:

Index = integer()

X = integer()

Y = integer()

Z = integer()

See *vertexAttrib1d/2*

vertexAttribI4i(Index, X, Y, Z, W) -> ok

Types:

Index = integer()

X = integer()

Y = integer()

Z = integer()

W = integer()

See *vertexAttrib1d/2*

vertexAttribI1ui(Index, X) -> ok

Types:

Index = integer()

X = integer()

See *vertexAttrib1d/2*

vertexAttribI2ui(Index, X, Y) -> ok

Types:

Index = integer()
X = integer()
Y = integer()

See *vertexAttrib1d/2*

vertexAttribI3ui(Index, X, Y, Z) -> ok

Types:

Index = integer()
X = integer()
Y = integer()
Z = integer()

See *vertexAttrib1d/2*

vertexAttribI4ui(Index, X, Y, Z, W) -> ok

Types:

Index = integer()
X = integer()
Y = integer()
Z = integer()
W = integer()

See *vertexAttrib1d/2*

vertexAttribI1iv(Index::integer(), V) -> ok

Types:

V = {X::integer()}

Equivalent to *vertexAttribI1i(Index, X)*.

vertexAttribI2iv(Index::integer(), V) -> ok

Types:

V = {X::integer(), Y::integer()}

Equivalent to *vertexAttribI2i(Index, X, Y)*.

vertexAttribI3iv(Index::integer(), V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer()}

Equivalent to *vertexAttribI3i(Index, X, Y, Z)*.

vertexAttribI4iv(Index::integer(), V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer(), W::integer()}

Equivalent to *vertexAttribI4i(Index, X, Y, Z, W)*.

vertexAttribI1uiv(Index::integer(), V) -> ok

Types:

V = {X::integer()}

Equivalent to *vertexAttribI1ui(Index, X)*.

vertexAttribI2uiv(Index::integer(), V) -> ok

Types:

V = {X::integer(), Y::integer()}

Equivalent to *vertexAttribI2ui(Index, X, Y)*.

vertexAttribI3uiv(Index::integer(), V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer()}

Equivalent to *vertexAttribI3ui(Index, X, Y, Z)*.

vertexAttribI4uiv(Index::integer(), V) -> ok

Types:

V = {X::integer(), Y::integer(), Z::integer(), W::integer()}

Equivalent to *vertexAttribI4ui(Index, X, Y, Z, W)*.

vertexAttribI4bv(Index, V) -> ok

Types:

Index = integer()

V = {integer(), integer(), integer(), integer()}

See *vertexAttrib1d/2*

vertexAttribI4sv(Index, V) -> ok

Types:

Index = integer()

V = {integer(), integer(), integer(), integer()}

See *vertexAttrib1d/2*

vertexAttribI4ubv(Index, V) -> ok

Types:

Index = integer()

V = {integer(), integer(), integer(), integer()}

See *vertexAttrib1d/2*

vertexAttribI4usv(Index, V) -> ok

Types:

Index = integer()

V = {integer(), integer(), integer(), integer()}

See *vertexAttrib1d/2*

getUniformuiv(Program, Location) -> {integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer()}

Types:

Program = integer()

Location = integer()

See *getUniformfv/2*

bindFragDataLocation(Program, Color, Name) -> ok

Types:

Program = integer()

Color = integer()

Name = string()

Bind a user-defined varying out variable to a fragment shader color number

`gl:bindFragDataLocation` explicitly specifies the binding of the user-defined varying out variable `Name` to fragment shader color number `ColorNumber` for program `Program` . If `Name` was bound previously, its assigned binding is replaced with `ColorNumber` . `Name` must be a null-terminated string. `ColorNumber` must be less than `?GL_MAX_DRAW_BUFFERS` .

The bindings specified by `gl:bindFragDataLocation` have no effect until `Program` is next linked. Bindings may be specified at any time after `Program` has been created. Specifically, they may be specified before shader objects are attached to the program. Therefore, any name may be specified in `Name` , including a name that is never used as a varying out variable in any fragment shader object. Names beginning with `?gl_` are reserved by the GL.

In addition to the errors generated by `gl:bindFragDataLocation`, the program `Program` will fail to link if:

The number of active outputs is greater than the value `?GL_MAX_DRAW_BUFFERS`.

More than one varying out variable is bound to the same color number.

See **external** documentation.

getFragDataLocation(Program, Name) -> integer()

Types:

Program = integer()

Name = string()

Query the bindings of color numbers to user-defined varying out variables

`gl:getFragDataLocation` retrieves the assigned color number binding for the user-defined varying out variable `Name` for program `Program` . `Program` must have previously been linked. `Name` must be a null-terminated string. If `Name` is not the name of an active user-defined varying out fragment shader variable within `Program` , -1 will be returned.

See **external** documentation.

uniform1ui(Location, V0) -> ok

Types:

Location = integer()

V0 = integer()

See *uniform1f/2*

uniform2ui(Location, V0, V1) -> ok

Types:

Location = integer()
V0 = integer()
V1 = integer()

See *uniform1f/2*

uniform3ui(Location, V0, V1, V2) -> ok

Types:

Location = integer()
V0 = integer()
V1 = integer()
V2 = integer()

See *uniform1f/2*

uniform4ui(Location, V0, V1, V2, V3) -> ok

Types:

Location = integer()
V0 = integer()
V1 = integer()
V2 = integer()
V3 = integer()

See *uniform1f/2*

uniform1uiv(Location, Value) -> ok

Types:

Location = integer()
Value = [integer()]

See *uniform1f/2*

uniform2uiv(Location, Value) -> ok

Types:

Location = integer()
Value = [{integer(), integer()}]

See *uniform1f/2*

uniform3uiv(Location, Value) -> ok

Types:

Location = integer()
Value = [{integer(), integer(), integer()}]

See *uniform1f/2*

uniform4uiv(Location, Value) -> ok

Types:

Location = integer()
Value = [{integer(), integer(), integer(), integer()}]

See *uniform1f/2*

texParameterIiv(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {integer()}

See *texParameterf/3*

texParameterIuiv(Target, Pname, Params) -> ok

Types:

Target = enum()
Pname = enum()
Params = {integer()}

glTexParameterI

See **external** documentation.

getTexParameterIiv(Target, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Target = enum()
Pname = enum()

See *getTexParameterfv/2*

getTexParameterIuiv(Target, Pname) -> {integer(), integer(), integer(), integer()}

Types:

Target = enum()
Pname = enum()

glGetTexParameterI

See **external** documentation.

clearBufferiv(Buffer, Drawbuffer, Value) -> ok

Types:

Buffer = enum()
Drawbuffer = integer()
Value = {integer()}

Clear individual buffers of the currently bound draw framebuffer

gl:clearBuffer* clears the specified buffer to the specified value(s). If Buffer is ?GL_COLOR, a particular draw buffer ?GL_DRAWBUFFER I is specified by passing I as DrawBuffer . In this case, Value points to a four-element vector specifying the R, G, B and A color to clear that draw buffer to. If Buffer is one of ?GL_FRONT, ?GL_BACK, ?GL_LEFT, ?GL_RIGHT, or ?GL_FRONT_AND_BACK , identifying multiple buffers, each selected buffer is cleared to the same value. Clamping and conversion for fixed-point color buffers are performed in the same fashion as *gl:clearColor/4* .

If Buffer is ?GL_DEPTH, DrawBuffer must be zero, and Value points to a single value to clear the depth buffer to. Only gl:clearBufferfv should be used to clear depth buffers. Clamping and conversion for fixed-point depth buffers are performed in the same fashion as *gl:clearDepth/1* .

If Buffer is ?GL_STENCIL, DrawBuffer must be zero, and Value points to a single value to clear the stencil buffer to. Only gl:clearBufferiv should be used to clear stencil buffers. Masing and type conversion are performed in the same fashion as *gl:clearStencil/1* .

gl:clearBufferfi may be used to clear the depth and stencil buffers. Buffer must be ?GL_DEPTH_STENCIL and DrawBuffer must be zero. Depth and Stencil are the depth and stencil values, respectively.

The result of gl:clearBuffer is undefined if no conversion between the type of Value and the buffer being cleared is defined. However, this is not an error.

See **external** documentation.

clearBufferuiv(Buffer, Drawbuffer, Value) -> ok

Types:

Buffer = enum()
Drawbuffer = integer()
Value = {integer()}

See *clearBufferiv/3*

clearBufferfv(Buffer, Drawbuffer, Value) -> ok

Types:

Buffer = enum()
Drawbuffer = integer()
Value = {float()}

See *clearBufferiv/3*

clearBufferfi(Buffer, Drawbuffer, Depth, Stencil) -> ok

Types:

Buffer = enum()
Drawbuffer = integer()
Depth = float()
Stencil = integer()

glClearBufferfi

See **external** documentation.

getStringi(Name, Index) -> string()

Types:

Name = enum()

```
Index = integer()
```

See *getString/1*

drawArraysInstanced(Mode, First, Count, Primcount) -> ok

Types:

```
Mode = enum()
First = integer()
Count = integer()
Primcount = integer()
```

glDrawArraysInstance

See **external** documentation.

drawElementsInstanced(Mode, Count, Type, Indices, Primcount) -> ok

Types:

```
Mode = enum()
Count = integer()
Type = enum()
Indices = offset() | mem()
Primcount = integer()
```

glDrawElementsInstance

See **external** documentation.

texBuffer(Target, Internalformat, Buffer) -> ok

Types:

```
Target = enum()
Internalformat = enum()
Buffer = integer()
```

Attach the storage for a buffer object to the active buffer texture

`gl:texBuffer` attaches the storage for the buffer object named `Buffer` to the active buffer texture, and specifies the internal format for the texel array found in the attached buffer object. If `Buffer` is zero, any buffer object attached to the buffer texture is detached and no new buffer object is attached. If `Buffer` is non-zero, it must be the name of an existing buffer object. `Target` must be `?GL_TEXTURE_BUFFER` . `Internalformat` specifies the storage format, and must be one of the following sized internal formats:

`Component`
`Sized Internal Format` `Base Type` `Components` `Norm` 0 1 2 3
`?GL_R8` ubyte 1 YES R 0 0 1
`?GL_R16` ushort 1 YES R 0 0 1
`?GL_R16F` half 1 NO R 0 0 1
`?GL_R32F` float 1 NO R 0 0 1
`?GL_R8I` byte 1 NO R 0 0 1
`?GL_R16I` short 1 NO R 0 0 1
`?GL_R32I` int 1 NO R 0 0 1
`?GL_R8UI` ubyte 1 NO R 0 0 1
`?GL_R16UI` ushort 1 NO R 0 0 1
`?GL_R32UI` uint 1 NO R 0 0 1

?GL_RG8ubyte 2YESRG01
?GL_RG16 ushort2YESRG01
?GL_RG16Fhalf2NORG0 1
?GL_RG32Ffloat2NORG 01
?GL_RG8Ibyte2NO RG01
?GL_RG16Ishort 2NORG01
?GL_RG32I int2NORG01
?GL_RG8UI ubyte2NORG01
?GL_RG16UIushort2NORG0 1
?GL_RG32UIuint2NORG 01
?GL_RGB32Ffloat3NO RGB1
?GL_RGB32Iint 3NORGB1
?GL_RGB32UI uint3NORGB1
?GL_RGBA8uint4YESRGB A
?GL_RGBA16short4YESR GBA
?GL_RGBA16Fhalf4NO RGBA
?GL_RGBA32Ffloat 4NORGBA
?GL_RGBA8I byte4NORGBA
?GL_RGBA16Ishort4NORGB A
?GL_RGBA32Iint4NORG BA
?GL_RGBA8UIubyte4NO RGBA
?GL_RGBA16UIushort 4NORGBA
?GL_RGBA32UI uint4NORGBA

When a buffer object is attached to a buffer texture, the buffer object's data store is taken as the texture's texel array. The number of texels in the buffer texture's texel array is given by buffer_size components× sizeof(base_type/)

where `buffer_size` is the size of the buffer object, in basic machine units and components and base type are the element count and base data type for elements, as specified in the table above. The number of texels in the texel array is then clamped to the implementation-dependent limit `?GL_MAX_TEXTURE_BUFFER_SIZE`. When a buffer texture is accessed in a shader, the results of a texel fetch are undefined if the specified texel coordinate is negative, or greater than or equal to the clamped number of texels in the texel array.

See **external** documentation.

`primitiveRestartIndex(Index) -> ok`

Types:

`Index = integer()`

Specify the primitive restart index

`gl:primitiveRestartIndex` specifies a vertex array element that is treated specially when primitive restarting is enabled. This is known as the primitive restart index.

When one of the `Draw*` commands transfers a set of generic attribute array elements to the GL, if the index within the vertex arrays corresponding to that set is equal to the primitive restart index, then the GL does not process those elements as a vertex. Instead, it is as if the drawing command ended with the immediately preceding transfer, and another drawing command is immediately started with the same parameters, but only transferring the immediately following element through the end of the originally specified elements.

When either *gl:drawElementsBaseVertex/5* , *gl:drawElementsInstancedBaseVertex/6* or see `glMultiDrawElementsBaseVertex` is used, the primitive restart comparison occurs before the basevertex offset is added to the array index.

See **external** documentation.

getInteger64i_v(Target, Index) -> [integer()]

Types:

Target = enum()

Index = integer()

See *getBooleanv/1*

getBufferParameteri64v(Target, Pname) -> [integer()]

Types:

Target = enum()

Pname = enum()

glGetBufferParameteri64v

See **external** documentation.

framebufferTexture(Target, Attachment, Texture, Level) -> ok

Types:

Target = enum()

Attachment = enum()

Texture = integer()

Level = integer()

Attach a level of a texture object as a logical buffer to the currently bound framebuffer object

`gl:framebufferTexture`, `gl:framebufferTexture1D`, `gl:framebufferTexture2D`, and `gl:framebufferTexture` attach a selected mipmap level or image of a texture object as one of the logical buffers of the framebuffer object currently bound to `Target` . `Target` must be `?GL_DRAW_FRAMEBUFFER`, `?GL_READ_FRAMEBUFFER`, or `?GL_FRAMEBUFFER` . `?GL_FRAMEBUFFER` is equivalent to `?GL_DRAW_FRAMEBUFFER`.

`Attachment` specifies the logical attachment of the framebuffer and must be `?GL_COLOR_ATTACHMENT i`, `?GL_DEPTH_ATTACHMENT`, `?GL_STENCIL_ATTACHMENT` or `?GL_DEPTH_STENCIL_ATTACHMMENT` . `i` in `?GL_COLOR_ATTACHMENTi` may range from zero to the value of `?GL_MAX_COLOR_ATTACHMENTS` - 1. Attaching a level of a texture to `?GL_DEPTH_STENCIL_ATTACHMENT` is equivalent to attaching that level to both the `?GL_DEPTH_ATTACHMENTand` the `?GL_STENCIL_ATTACHMENT` attachment points simultaneously.

`Textarget` specifies what type of texture is named by `Texture` , and for cube map textures, specifies the face that is to be attached. If `Texture` is not zero, it must be the name of an existing texture with type `Textarget` , unless it is a cube map texture, in which case `Textarget` must be `?GL_TEXTURE_CUBE_MAP_POSITIVE_X` `?GL_TEXTURE_CUBE_MAP_NEGATIVE_X`, `?GL_TEXTURE_CUBE_MAP_POSITIVE_Y`, `?GL_TEXTURE_CUBE_MAP_NEGATIVE_Y` , `?GL_TEXTURE_CUBE_MAP_POSITIVE_Z`, or `?GL_TEXTURE_CUBE_MAP_NEGATIVE_Z`.

If `Texture` is non-zero, the specified `Level` of the texture object named `Texture` is attached to the framebufer attachment point named by `Attachment` . For `gl:framebufferTexture1D` , `gl:framebufferTexture2D`, and `gl:framebufferTexture3D`, `Texture` must be zero or the name of an existing texture with a target of `Textarget` , or `Texture` must be the name of an existing cube-map texture and `Textarget` must be one of `?GL_TEXTURE_CUBE_MAP_POSITIVE_X` , `?GL_TEXTURE_CUBE_MAP_POSITIVE_Y`, `?GL_TEXTURE_CUBE_MAP_POSITIVE_Z`, `?GL_TEXTURE_CUBE_MAP_NEGATIVE_X` , `?GL_TEXTURE_CUBE_MAP_NEGATIVE_Y`, or `?GL_TEXTURE_CUBE_MAP_NEGATIVE_Z`.

If Textarget is ?GL_TEXTURE_RECTANGLE, ?GL_TEXTURE_2D_MULTISAMPLE, or ?GL_TEXTURE_2D_MULTISAMPLE_ARRAY, then Level must be zero. If Textarget is ?GL_TEXTURE_3D, then level must be greater than or equal to zero and less than or equal to log2 of the value of ?GL_MAX_3D_TEXTURE_SIZE. If Textarget is one of ?GL_TEXTURE_CUBE_MAP_POSITIVE_X, ?GL_TEXTURE_CUBE_MAP_POSITIVE_Y, ?GL_TEXTURE_CUBE_MAP_POSITIVE_Z , ?GL_TEXTURE_CUBE_MAP_NEGATIVE_X, ?GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, or ?GL_TEXTURE_CUBE_MAP_NEGATIVE_Z , then Level must be greater than or equal to zero and less than or equal to log2 of the value of ?GL_MAX_CUBE_MAP_TEXTURE_SIZE. For all other values of Textarget , Level must be greater than or equal to zero and no larger than log2 of the value of ?GL_MAX_TEXTURE_SIZE.

Layer specifies the layer of a 2-dimensional image within a 3-dimensional texture.

For gl:framebufferTexture1D, if Texture is not zero, then Textarget must be ?GL_TEXTURE_1D. For gl:framebufferTexture2D, if Texture is not zero, Textarget must be one of ?GL_TEXTURE_2D, ?GL_TEXTURE_RECTANGLE , ?GL_TEXTURE_CUBE_MAP_POSITIVE_X, ?GL_TEXTURE_CUBE_MAP_POSITIVE_Y, ?GL_TEXTURE_CUBE_MAP_POSITIVE_Z , ?GL_TEXTURE_CUBE_MAP_NEGATIVE_X, ?GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, ?GL_TEXTURE_CUBE_MAP_NEGATIVE_Z , or ?GL_TEXTURE_2D_MULTISAMPLE. For gl:framebufferTexture3D, if Texture is not zero, then Textarget must be ?GL_TEXTURE_3D.

See **external** documentation.

vertexAttribDivisor(Index, Divisor) -> ok

Types:

Index = integer()
Divisor = integer()

Modify the rate at which generic vertex attributes advance during instanced rendering

gl:vertexAttribDivisor modifies the rate at which generic vertex attributes advance when rendering multiple instances of primitives in a single draw call. If Divisor is zero, the attribute at slot Index advances once per vertex. If Divisor is non-zero, the attribute advances once per Divisor instances of the set(s) of vertices being rendered. An attribute is referred to as instanced if its ?GL_VERTEX_ATTRIB_ARRAY_DIVISOR value is non-zero.

Index must be less than the value of ?GL_MAX_VERTEX_ATTRIBUTES.

See **external** documentation.

minSampleShading(Value) -> ok

Types:

Value = clamp()

Specifies minimum rate at which sample shaing takes place

gl:minSampleShading specifies the rate at which samples are shaded within a covered pixel. Sample-rate shading is enabled by calling *gl:enable/1* with the parameter ?GL_SAMPLE_SHADING . If ?GL_MULTISAMPLE or ?GL_SAMPLE_SHADING is disabled, sample shading has no effect. Otherwise, an implementation must provide at least as many unique color values for each covered fragment as specified by Value times Samples where Samples is the value of ?GL_SAMPLES for the current framebuffer. At least 1 sample for each covered fragment is generated.

A Value of 1.0 indicates that each sample in the framebuffer should be indpendently shaded. A Value of 0.0 effectively allows the GL to ignore sample rate shading. Any value between 0.0 and 1.0 allows the GL to shade only a subset of the total samples within each covered fragment. Which samples are shaded and the algorithm used to select that subset of the fragment's samples is implementation dependent.

See **external** documentation.

blendEquationi(Buf, Mode) -> ok

Types:

Buf = integer()

Mode = enum()

See *blendEquation/1*

blendEquationSeparatei(Buf, ModeRGB, ModeAlpha) -> ok

Types:

Buf = integer()

ModeRGB = enum()

ModeAlpha = enum()

See *blendEquationSeparate/2*

blendFunci(Buf, Src, Dst) -> ok

Types:

Buf = integer()

Src = enum()

Dst = enum()

glBlendFunci

See **external** documentation.

blendFuncSeparatei(Buf, SrcRGB, DstRGB, SrcAlpha, DstAlpha) -> ok

Types:

Buf = integer()

SrcRGB = enum()

DstRGB = enum()

SrcAlpha = enum()

DstAlpha = enum()

See *blendFuncSeparate/4*

loadTransposeMatrixfARB(M) -> ok

Types:

M = matrix()

glLoadTransposeMatrixARB

See **external** documentation.

loadTransposeMatrixdARB(M) -> ok

Types:

M = matrix()

glLoadTransposeMatrixARB

See **external** documentation.

multTransposeMatrixfARB(M) -> ok

Types:

M = matrix()

glMultTransposeMatrixARB

See **external** documentation.

multTransposeMatrixdARB(M) -> ok

Types:

M = matrix()

glMultTransposeMatrixARB

See **external** documentation.

weightbvARB(Weights) -> ok

Types:

Weights = [integer()]

glWeightARB

See **external** documentation.

weightsvARB(Weights) -> ok

Types:

Weights = [integer()]

glWeightARB

See **external** documentation.

weightivARB(Weights) -> ok

Types:

Weights = [integer()]

glWeightARB

See **external** documentation.

weightfvARB(Weights) -> ok

Types:

Weights = [float()]

glWeightARB

See **external** documentation.

weightdvARB(Weights) -> ok

Types:

Weights = [float()]

glWeightARB

See **external** documentation.

weightubvARB(Weights) -> ok

Types:

Weights = [integer()]

glWeightARB

See **external** documentation.

weightusvARB(Weights) -> ok

Types:

Weights = [integer()]

glWeightARB

See **external** documentation.

weightuivARB(Weights) -> ok

Types:

Weights = [integer()]

glWeightARB

See **external** documentation.

vertexBlendARB(Count) -> ok

Types:

Count = integer()

glVertexBlenARB

See **external** documentation.

currentPaletteMatrixARB(Index) -> ok

Types:

Index = integer()

glCurrentPaletteMatrixARB

See **external** documentation.

matrixIndexubvARB(Indices) -> ok

Types:

Indices = [integer()]

glMatrixIndexARB

See **external** documentation.

matrixIndexusvARB(Indices) -> ok

Types:

Indices = [integer()]

glMatrixIndexARB

See **external** documentation.

matrixIndexuivARB(Indices) -> ok

Types:

Indices = [integer()]

glMatrixIndexARB

See **external** documentation.

programStringARB(Target, Format, String) -> ok

Types:

Target = enum()
Format = enum()
String = string()

glProgramStringARB

See **external** documentation.

bindProgramARB(Target, Program) -> ok

Types:

Target = enum()
Program = integer()

glBindProgramARB

See **external** documentation.

deleteProgramsARB(Programs) -> ok

Types:

Programs = [integer()]

glDeleteProgramsARB

See **external** documentation.

genProgramsARB(N) -> [integer()]

Types:

N = integer()

glGenProgramsARB

See **external** documentation.

programEnvParameter4dARB(Target, Index, X, Y, Z, W) -> ok

Types:

Target = enum()
Index = integer()
X = float()
Y = float()
Z = float()

```
W = float()
```

glProgramEnvParameterARB

See **external** documentation.

programEnvParameter4dvARB(Target, Index, Params) -> ok

Types:

```
Target = enum()
Index = integer()
Params = {float(), float(), float(), float()}
```

glProgramEnvParameterARB

See **external** documentation.

programEnvParameter4fARB(Target, Index, X, Y, Z, W) -> ok

Types:

```
Target = enum()
Index = integer()
X = float()
Y = float()
Z = float()
W = float()
```

glProgramEnvParameterARB

See **external** documentation.

programEnvParameter4fvARB(Target, Index, Params) -> ok

Types:

```
Target = enum()
Index = integer()
Params = {float(), float(), float(), float()}
```

glProgramEnvParameterARB

See **external** documentation.

programLocalParameter4dARB(Target, Index, X, Y, Z, W) -> ok

Types:

```
Target = enum()
Index = integer()
X = float()
Y = float()
Z = float()
W = float()
```

glProgramLocalParameterARB

See **external** documentation.

programLocalParameter4dvARB(Target, Index, Params) -> ok

Types:

```
Target = enum()
Index = integer()
Params = {float(), float(), float(), float()}
```

glProgramLocalParameterARB

See **external** documentation.

programLocalParameter4fARB(Target, Index, X, Y, Z, W) -> ok

Types:

```
Target = enum()
Index = integer()
X = float()
Y = float()
Z = float()
W = float()
```

glProgramLocalParameterARB

See **external** documentation.

programLocalParameter4fvARB(Target, Index, Params) -> ok

Types:

```
Target = enum()
Index = integer()
Params = {float(), float(), float(), float()}
```

glProgramLocalParameterARB

See **external** documentation.

getProgramEnvParameterdvARB(Target, Index) -> {float(), float(), float(), float()}

Types:

```
Target = enum()
Index = integer()
```

glGetProgramEnvParameterARB

See **external** documentation.

getProgramEnvParameterfvARB(Target, Index) -> {float(), float(), float(), float()}

Types:

```
Target = enum()
Index = integer()
```

glGetProgramEnvParameterARB

See **external** documentation.

getProgramLocalParameterdvARB(Target, Index) -> {float(), float(), float(), float()}

Types:

Target = enum()

Index = integer()

glGetProgramLocalParameterARB

See **external** documentation.

getProgramLocalParameterfvARB(Target, Index) -> {float(), float(), float(), float()}

Types:

Target = enum()

Index = integer()

glGetProgramLocalParameterARB

See **external** documentation.

getProgramStringARB(Target, Pname, String) -> ok

Types:

Target = enum()

Pname = enum()

String = mem()

glGetProgramStringARB

See **external** documentation.

getBufferParameterivARB(Target, Pname) -> [integer()]

Types:

Target = enum()

Pname = enum()

glGetBufferParameterARB

See **external** documentation.

deleteObjectARB(Obj) -> ok

Types:

Obj = integer()

glDeleteObjectARB

See **external** documentation.

getHandleARB(Pname) -> integer()

Types:

Pname = enum()

glGetHandleARB

See **external** documentation.

detachObjectARB(ContainerObj, AttachedObj) -> ok

Types:

ContainerObj = integer()

AttachedObj = integer()

glDetachObjectARB

See **external** documentation.

createShaderObjectARB(ShaderType) -> integer()

Types:

ShaderType = enum()

glCreateShaderObjectARB

See **external** documentation.

shaderSourceARB(ShaderObj, String) -> ok

Types:

ShaderObj = integer()

String = [string()]

glShaderSourceARB

See **external** documentation.

compileShaderARB(ShaderObj) -> ok

Types:

ShaderObj = integer()

glCompileShaderARB

See **external** documentation.

createProgramObjectARB() -> integer()

glCreateProgramObjectARB

See **external** documentation.

attachObjectARB(ContainerObj, Obj) -> ok

Types:

ContainerObj = integer()

Obj = integer()

glAttachObjectARB

See **external** documentation.

linkProgramARB(ProgramObj) -> ok

Types:

ProgramObj = integer()

glLinkProgramARB

See **external** documentation.

useProgramObjectARB(ProgramObj) -> ok

Types:

ProgramObj = integer()

glUseProgramObjectARB

See **external** documentation.

validateProgramARB(ProgramObj) -> ok

Types:

ProgramObj = integer()

glValidateProgramARB

See **external** documentation.

getObjectParameterfvARB(Obj, Pname) -> float()

Types:

Obj = integer()

Pname = enum()

glGetObjectParameterARB

See **external** documentation.

getObjectParameterivARB(Obj, Pname) -> integer()

Types:

Obj = integer()

Pname = enum()

glGetObjectParameterARB

See **external** documentation.

getInfoLogARB(Obj, MaxLength) -> string()

Types:

Obj = integer()

MaxLength = integer()

glGetInfoLogARB

See **external** documentation.

getAttachedObjectsARB(ContainerObj, MaxCount) -> [integer()]

Types:

ContainerObj = integer()

MaxCount = integer()

glGetAttachedObjectsARB

See **external** documentation.

getUniformLocationARB(ProgramObj, Name) -> integer()

Types:

ProgramObj = integer()

Name = string()

glGetUniformLocationARB

See **external** documentation.

getActiveUniformARB(ProgramObj, Index, MaxLength) -> {Size::integer(), Type::enum(), Name::string()}

Types:

ProgramObj = integer()

Index = integer()

MaxLength = integer()

glGetActiveUniformARB

See **external** documentation.

getUniformfvARB(ProgramObj, Location) -> matrix()

Types:

ProgramObj = integer()

Location = integer()

glGetUniformARB

See **external** documentation.

getUniformivARB(ProgramObj, Location) -> {integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer()}

Types:

ProgramObj = integer()

Location = integer()

glGetUniformARB

See **external** documentation.

getShaderSourceARB(Obj, MaxLength) -> string()

Types:

Obj = integer()

MaxLength = integer()

glGetShaderSourceARB

See **external** documentation.

bindAttribLocationARB(ProgramObj, Index, Name) -> ok

Types:

ProgramObj = integer()

Index = integer()

Name = string()

glBindAttribLocationARB

See **external** documentation.

getActiveAttribARB(ProgramObj, Index, MaxLength) -> {Size::integer(), Type::enum(), Name::string()}

Types:

ProgramObj = integer()

Index = integer()

MaxLength = integer()

glGetActiveAttribARB

See **external** documentation.

getAttribLocationARB(ProgramObj, Name) -> integer()

Types:

ProgramObj = integer()

Name = string()

glGetAttribLocationARB

See **external** documentation.

isRenderbuffer(Renderbuffer) -> 0 | 1

Types:

Renderbuffer = integer()

Determine if a name corresponds to a renderbuffer object

`gl:isRenderbuffer` returns `?GL_TRUE` if `Renderbuffer` is currently the name of a renderbuffer object. If `Renderbuffer` is zero, or if `Renderbuffer` is not the name of a renderbuffer object, or if an error occurs, `gl:isRenderbuffer` returns `?GL_FALSE`. If `Renderbuffer` is a name returned by *gl:genRenderbuffers/1* , by that has not yet been bound through a call to *gl:bindRenderbuffer/2* or *gl:framebufferRenderbuffer/4* , then the name is not a renderbuffer object and `gl:isRenderbuffer` returns `?GL_FALSE` .

See **external** documentation.

bindRenderbuffer(Target, Renderbuffer) -> ok

Types:

Target = enum()

Renderbuffer = integer()

Bind a renderbuffer to a renderbuffer target

`gl:bindRenderbuffer` binds the renderbuffer object with name `Renderbuffer` to the renderbuffer target specified by `Target` . `Target` must be `?GL_RENDERBUFFER` . `Renderbuffer` is the name of a renderbuffer object previously returned from a call to *gl:genRenderbuffers/1* , or zero to break the existing binding of a renderbuffer object to `Target` .

See **external** documentation.

deleteRenderbuffers(Renderbuffers) -> ok

Types:

Renderbuffers = [integer()]

Delete renderbuffer objects

`gl:deleteRenderbuffers` deletes the `N` renderbuffer objects whose names are stored in the array addressed by `Renderbuffers` . The name zero is reserved by the GL and is silently ignored, should it occur in `Renderbuffers` , as are other unused names. Once a renderbuffer object is deleted, its name is again unused and it has no contents. If a renderbuffer that is currently bound to the target `?GL_RENDERBUFFER` is deleted, it is as though *gl:bindRenderbuffer/2* had been executed with a `Target` of `?GL_RENDERBUFFER` and a `Name` of zero.

If a renderbuffer object is attached to one or more attachment points in the currently bound framebuffer, then it as if *gl:framebufferRenderbuffer/4* had been called, with a `Renderbuffer` of zero for each attachment point to which this image was attached in the currently bound framebuffer. In other words, this renderbuffer object is first detached from all attachment ponits in the currently bound framebuffer. Note that the renderbuffer image is specifically `not` detached from any non-bound framebuffers.

See **external** documentation.

genRenderbuffers(N) -> [integer()]

Types:

N = integer()

Generate renderbuffer object names

`gl:genRenderbuffers` returns `N` renderbuffer object names in `Renderbuffers` . There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to `gl:genRenderbuffers` .

Renderbuffer object names returned by a call to `gl:genRenderbuffers` are not returned by subsequent calls, unless they are first deleted with *gl:deleteRenderbuffers/1* .

The names returned in `Renderbuffers` are marked as used, for the purposes of `gl:genRenderbuffers` only, but they acquire state and type only when they are first bound.

See **external** documentation.

renderbufferStorage(Target, Internalformat, Width, Height) -> ok

Types:

Target = enum()

Internalformat = enum()

Width = integer()

Height = integer()

Establish data storage, format and dimensions of a renderbuffer object's image

`gl:renderbufferStorage` is equivalent to calling *gl:renderbufferStorageMultisample/5* with the `Samples` set to zero.

The target of the operation, specified by `Target` must be `?GL_RENDERBUFFER`. `Internalformat` specifies the internal format to be used for the renderbuffer object's storage and must be a color-renderable, depth-renderable, or stencil-renderable format. `Width` and `Height` are the dimensions, in pixels, of the renderbuffer. Both `Width` and `Height` must be less than or equal to the value of `?GL_MAX_RENDERBUFFER_SIZE` .

Upon success, `gl:renderbufferStorage` deletes any existing data store for the renderbuffer image and the contents of the data store after calling `gl:renderbufferStorage` are undefined.

See **external** documentation.

getRenderbufferParameteriv(Target, Pname) -> integer()

Types:

Target = enum()

Pname = enum()

Retrieve information about a bound renderbuffer object

`gl:getRenderbufferParameteriv` retrieves information about a bound renderbuffer object. `Target` specifies the target of the query operation and must be `?GL_RENDERBUFFER` . `Pname` specifies the parameter whose value to query and must be one of `?GL_RENDERBUFFER_WIDTH` , `?GL_RENDERBUFFER_HEIGHT`, `?GL_RENDERBUFFER_INTERNAL_FORMAT`, `?GL_RENDERBUFFER_RED_SIZE` , `?GL_RENDERBUFFER_GREEN_SIZE`, `?GL_RENDERBUFFER_BLUE_SIZE`, `?GL_RENDERBUFFER_ALPHA_SIZE` , `?GL_RENDERBUFFER_DEPTH_SIZE`, `?GL_RENDERBUFFER_DEPTH_SIZE`, `?GL_RENDERBUFFER_STENCIL_SIZE` , or `?GL_RENDERBUFFER_SAMPLES`.

Upon a successful return from `gl:getRenderbufferParameteriv`, if `Pname` is `?GL_RENDERBUFFER_WIDTH` , `?GL_RENDERBUFFER_HEIGHT`, `?GL_RENDERBUFFER_INTERNAL_FORMAT`, or `?GL_RENDERBUFFER_SAMPLES` , then `Params` will contain the width in pixels, the height in pixels, the internal format, or the number of samples, respectively, of the image of the renderbuffer currently bound to `Target` .

If `Pname` is `?GL_RENDERBUFFER_RED_SIZE`, `?GL_RENDERBUFFER_GREEN_SIZE`, `?GL_RENDERBUFFER_BLUE_SIZE`, `?GL_RENDERBUFFER_ALPHA_SIZE`, `?GL_RENDERBUFFER_DEPTH_SIZE` , or `?GL_RENDERBUFFER_STENCIL_SIZE`, then `Params` will contain the actual resolutions (not the resolutions specified when the image array was defined) for the red, green, blue, alpha depth, or stencil components, respectively, of the image of the renderbuffer currently bound to `Target` .

See **external** documentation.

isFramebuffer(Framebuffer) -> 0 | 1

Types:

Framebuffer = integer()

Determine if a name corresponds to a framebuffer object

`gl:isFramebuffer` returns `?GL_TRUE` if `Framebuffer` is currently the name of a framebuffer object. If `Framebuffer` is zero, or if `?framebuffer` is not the name of a framebuffer object, or if an error occurs, `gl:isFramebuffer` returns `?GL_FALSE`. If `Framebuffer` is a name returned by *gl:genFramebuffers/1* , by that has not yet been bound through a call to *gl:bindFramebuffer/2* , then the name is not a framebuffer object and `gl:isFramebuffer` returns `?GL_FALSE`.

See **external** documentation.

bindFramebuffer(Target, Framebuffer) -> ok

Types:

Target = enum()

Framebuffer = integer()

Bind a framebuffer to a framebuffer target

`gl:bindFramebuffer` binds the framebuffer object with name `Framebuffer` to the framebuffer target specified by `Target` . `Target` must be either `?GL_DRAW_FRAMEBUFFER` , `?GL_READ_FRAMEBUFFER`

or ?GL_FRAMEBUFFER. If a framebuffer object is bound to ?GL_DRAW_FRAMEBUFFER or ?GL_READ_FRAMEBUFFER, it becomes the target for rendering or readback operations, respectively, until it is deleted or another framebuffer is bound to the corresponding bind point. Calling gl:bindFramebuffer with Target set to ?GL_FRAMEBUFFER binds Framebuffer to both the read and draw framebuffer targets. Framebuffer is the name of a framebuffer object previously returned from a call to *gl:genFramebuffers/1* , or zero to break the existing binding of a framebuffer object to Target .

See **external** documentation.

deleteFramebuffers(Framebuffers) -> ok

Types:

Framebuffers = [integer()]

Delete framebuffer objects

gl:deleteFramebuffers deletes the N framebuffer objects whose names are stored in the array addressed by Framebuffers . The name zero is reserved by the GL and is silently ignored, should it occur in Framebuffers , as are other unused names. Once a framebuffer object is deleted, its name is again unused and it has no attachments. If a framebuffer that is currently bound to one or more of the targets ?GL_DRAW_FRAMEBUFFER or ?GL_READ_FRAMEBUFFER is deleted, it is as though *gl:bindFramebuffer/2* had been executed with the corresponding Target and Framebuffer zero.

See **external** documentation.

genFramebuffers(N) -> [integer()]

Types:

N = integer()

Generate framebuffer object names

gl:genFramebuffers returns N framebuffer object names in Ids . There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to gl:genFramebuffers .

Framebuffer object names returned by a call to gl:genFramebuffers are not returned by subsequent calls, unless they are first deleted with *gl:deleteFramebuffers/1* .

The names returned in Ids are marked as used, for the purposes of gl:genFramebuffers only, but they acquire state and type only when they are first bound.

See **external** documentation.

checkFramebufferStatus(Target) -> enum()

Types:

Target = enum()

Check the completeness status of a framebuffer

gl:checkFramebufferStatus queries the completeness status of the framebuffer object currently bound to Target . Target must be ?GL_DRAW_FRAMEBUFFER, ?GL_READ_FRAMEBUFFER or ?GL_FRAMEBUFFER. ?GL_FRAMEBUFFER is equivalent to ?GL_DRAW_FRAMEBUFFER .

The return value is ?GL_FRAMEBUFFER_COMPLETE if the framebuffer bound to Target is complete. Otherwise, the return value is determined as follows:

?GL_FRAMEBUFFER_UNDEFINED is returned if Target is the default framebuffer, but the default framebuffer does not exist.

?GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT is returned if any of the framebuffer attachment points are framebuffer incomplete.

?GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT is returned if the framebuffer does not have at least one image attached to it.

?GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER is returned if the value of ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE is ?GL_NONE for any color attachment point(s) named by ?GL_DRAWBUFFERi.

?GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER is returned if ?GL_READ_BUFFER is not ?GL_NONE and the value of ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE is ?GL_NONE for the color attachment point named by ?GL_READ_BUFFER.

?GL_FRAMEBUFFER_UNSUPPORTED is returned if the combination of internal formats of the attached images violates an implementation-dependent set of restrictions.

?GL_FRAMEBUFFER_INCOMPLETE_MULTISAMPLE is returned if the value of ?GL_RENDERBUFFER_SAMPLES is not the same for all attached renderbuffers; if the value of ?GL_TEXTURE_SAMPLES is the not same for all attached textures; or, if the attached images are a mix of renderbuffers and textures, the value of ?GL_RENDERBUFFER_SAMPLES does not match the value of ?GL_TEXTURE_SAMPLES .

?GL_FRAMEBUFFER_INCOMPLETE_MULTISAMPLE is also returned if the value of ?GL_TEXTURE_FIXED_SAMPLE_LOCATIONS is not the same for all attached textures; or, if the attached images are a mix of renderbuffers and textures, the value of ?GL_TEXTURE_FIXED_SAMPLE_LOCATIONS is not ?GL_TRUE for all attached textures.

?GL_FRAMEBUFFER_INCOMPLETE_LAYER_TARGETS is returned if any framebuffer attachment is layered, and any populated attachment is not layered, or if all populated color attachments are not from textures of the same target.

Additionally, if an error occurs, zero is returned.

See **external** documentation.

framebufferTexture1D(Target, Attachment, Textarget, Texture, Level) -> ok

Types:

```
Target = enum()
Attachment = enum()
Textarget = enum()
Texture = integer()
Level = integer()
```

See *framebufferTexture/4*

framebufferTexture2D(Target, Attachment, Textarget, Texture, Level) -> ok

Types:

```
Target = enum()
Attachment = enum()
Textarget = enum()
Texture = integer()
Level = integer()
```

See *framebufferTexture/4*

framebufferTexture3D(Target, Attachment, Textarget, Texture, Level, Zoffset) -> ok

Types:

Target = enum()
Attachment = enum()
Textarget = enum()
Texture = integer()
Level = integer()
Zoffset = integer()

See *framebufferTexture/4*

framebufferRenderbuffer(Target, Attachment, Renderbuffertarget, Renderbuffer) -> ok

Types:

Target = enum()
Attachment = enum()
Renderbuffertarget = enum()
Renderbuffer = integer()

Attach a renderbuffer as a logical buffer to the currently bound framebuffer object

`gl:framebufferRenderbuffer` attaches a renderbuffer as one of the logical buffers of the currently bound framebuffer object. `Renderbuffer` is the name of the renderbuffer object to attach and must be either zero, or the name of an existing renderbuffer object of type `Renderbuffertarget` . If `Renderbuffer` is not zero and if `gl:framebufferRenderbuffer` is successful, then the renderbuffer name `Renderbuffer` will be used as the logical buffer identified by `Attachment` of the framebuffer currently bound to `Target` .

The value of `?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE` for the specified attachment point is set to `?GL_RENDERBUFFER` and the value of `?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_NAME` is set to `Renderbuffer` . All other state values of the attachment point specified by `Attachment` are set to their default values. No change is made to the state of the renderbuuffer object and any previous attachment to the `Attachment` logical buffer of the framebuffer `Target` is broken.

Calling `gl:framebufferRenderbuffer` with the renderbuffer name zero will detach the image, if any, identified by `Attachment` , in the framebuffer currently bound to `Target` . All state values of the attachment point specified by attachment in the object bound to target are set to their default values.

Setting `Attachment` to the value `?GL_DEPTH_STENCIL_ATTACHMENT` is a special case causing both the depth and stencil attachments of the framebuffer object to be set to `Renderbuffer` , which should have the base internal format `?GL_DEPTH_STENCIL` .

See **external** documentation.

getFramebufferAttachmentParameteriv(Target, Attachment, Pname) -> integer()

Types:

Target = enum()
Attachment = enum()
Pname = enum()

Retrieve information about attachments of a bound framebuffer object

gl:getFramebufferAttachmentParameter returns information about attachments of a bound framebuffer object. Target specifies the framebuffer binding point and must be ?GL_DRAW_FRAMEBUFFER, ?GL_READ_FRAMEBUFFER or ?GL_FRAMEBUFFER. ?GL_FRAMEBUFFER is equivalent to ?GL_DRAW_FRAMEBUFFER.

If the default framebuffer is bound to Target then Attachment must be one of ?GL_FRONT_LEFT, ?GL_FRONT_RIGHT, ?GL_BACK_LEFT, or ?GL_BACK_RIGHT , identifying a color buffer, ?GL_DEPTH, identifying the depth buffer, or ?GL_STENCIL , identifying the stencil buffer.

If a framebuffer object is bound, then Attachment must be one of ?GL_COLOR_ATTACHMENT i, ?GL_DEPTH_ATTACHMENT, ?GL_STENCIL_ATTACHMENT, or ?GL_DEPTH_STENCIL_ATTACHMENT . i in ?GL_COLOR_ATTACHMENTi must be in the range zero to the value of ?GL_MAX_COLOR_ATTACHMENTS - 1.

If Attachment is ?GL_DEPTH_STENCIL_ATTACHMENT and different objects are bound to the depth and stencil attachment points of Target the query will fail. If the same object is bound to both attachment points, information about that object will be returned.

Upon successful return from gl:getFramebufferAttachmentParameteriv, if Pname is ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE, then Params will contain one of ?GL_NONE , ?GL_FRAMEBUFFER_DEFAULT, ?GL_TEXTURE, or ?GL_RENDERBUFFER, identifying the type of object which contains the attached image. Other values accepted for Pname depend on the type of object, as described below.

If the value of ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE is ?GL_NONE, no framebuffer is bound to Target . In this case querying Pname ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_NAME will return zero, and all other queries will generate an error.

If the value of ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE is not ?GL_NONE, these queries apply to all other framebuffer types:

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_RED_SIZE, ?GL_FRAMEBUFFER_ATTACHMENT_GREEN_SIZE , ?GL_FRAMEBUFFER_ATTACHMENT_BLUE_SIZE, ?GL_FRAMEBUFFER_ATTACHMENT_ALPHA_SIZE , ?GL_FRAMEBUFFER_ATTACHMENT_DEPTH_SIZE, or ?GL_FRAMEBUFFER_ATTACHMENT_STENCIL_SIZE , then Params will contain the number of bits in the corresponding red, green, blue, alpha, depth, or stencil component of the specified attachment. Zero is returned if the requested component is not present in Attachment .

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_COMPONENT_TYPE, Params will contain the format of components of the specified attachment, one of ?GL_FLOAT, GL_INT , GL_UNSIGNED_INT , GL_SIGNED_NORMALIZED , or GL_UNSIGNED_NORMALIZED for floating-point, signed integer, unsigned integer, signed normalized fixed-point, or unsigned normalized fixed-point components respectively. Only color buffers may have integer components.

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_COLOR_ENCODING, Param will contain the encoding of components of the specified attachment, one of ?GL_LINEAR or ?GL_SRGB for linear or sRGB-encoded components, respectively. Only color buffer components may be sRGB-encoded; such components are treated as described in sections 4.1.7 and 4.1.8. For the default framebuffer, color encoding is determined by the implementation. For framebuffer objects, components are sRGB-encoded if the internal format of a color attachment is one of the color-renderable SRGB formats.

If the value of ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE is ?GL_RENDERBUFFER, then:

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_NAME, Params will contain the name of the renderbuffer object which contains the attached image.

If the value of ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE is ?GL_TEXTURE, then:

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_NAME, then Params will contain the name of the texture object which contains the attached image.

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_TEXTURE_LEVEL, then Params will contain the mipmap level of the texture object which contains the attached image.

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_TEXTURE_CUBE_MAP_FACE and the texture object named ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_NAME is a cube map texture, then Params will contain the cube map face of the cubemap texture object which contains the attached image. Otherwise Params will contain the value zero.

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_TEXTURE_LAYER and the texture object named ?GL_FRAMEBUFFER_ATTACHMENT_OBJECT_NAME is a layer of a three-dimensional texture or a one-or two-dimensional array texture, then Params will contain the number of the texture layer which contains the attached image. Otherwise Params will contain the value zero.

If Pname is ?GL_FRAMEBUFFER_ATTACHMENT_LAYERED, then Params will contain ?GL_TRUE if an entire level of a three-dimesional texture, cube map texture, or one-or two-dimensional array texture is attached. Otherwise, Params will contain ?GL_FALSE.

Any combinations of framebuffer type and Pname not described above will generate an error.

See **external** documentation.

generateMipmap(Target) -> ok

Types:

Target = enum()

Generate mipmaps for a specified texture target

gl:generateMipmap generates mipmaps for the texture attached to Target of the active texture unit. For cube map textures, a ?GL_INVALID_OPERATION error is generated if the texture attached to Target is not cube complete.

Mipmap generation replaces texel array levels level base+1 through q with arrays derived from the level base array, regardless of their previous contents. All other mimap arrays, including the level base array, are left unchanged by this computation.

The internal formats of the derived mipmap arrays all match those of the level base array. The contents of the derived arrays are computed by repeated, filtered reduction of the level base array. For one- and two-dimensional texture arrays, each layer is filtered independently.

See **external** documentation.

blitFramebuffer(SrcX0, SrcY0, SrcX1, SrcY1, DstX0, DstY0, DstX1, DstY1, Mask, Filter) -> ok

Types:

SrcX0 = integer()
SrcY0 = integer()
SrcX1 = integer()
SrcY1 = integer()
DstX0 = integer()
DstY0 = integer()
DstX1 = integer()
DstY1 = integer()
Mask = integer()
Filter = enum()

Copy a block of pixels from the read framebuffer to the draw framebuffer

gl:blitFramebuffer transfers a rectangle of pixel values from one region of the read framebuffer to another region in the draw framebuffer. Mask is the bitwise OR of a number of values indicating which buffers are to be copied. The values are ?GL_COLOR_BUFFER_BIT , ?GL_DEPTH_BUFFER_BIT, and ?GL_STENCIL_BUFFER_BIT. The pixels corresponding to these buffers are copied from the source rectangle bounded by the locations (SrcX0 ; SrcY0) and (SrcX1 ; SrcY1) to the destination rectangle bounded by the locations (DstX0 ; DstY0) and (DstX1 ; DstY1). The lower bounds of the rectangle are inclusive, while the upper bounds are exclusive.

The actual region taken from the read framebuffer is limited to the intersection of the source buffers being transferred, which may include the color buffer selected by the read buffer, the depth buffer, and/or the stencil buffer depending on mask. The actual region written to the draw framebuffer is limited to the intersection of the destination buffers being written, which may include multiple draw buffers, the depth buffer, and/or the stencil buffer depending on mask. Whether or not the source or destination regions are altered due to these limits, the scaling and offset applied to pixels being transferred is performed as though no such limits were present.

If the sizes of the source and destination rectangles are not equal, Filter specifies the interpolation method that will be applied to resize the source image , and must be ?GL_NEAREST or ?GL_LINEAR. ?GL_LINEAR is only a valid interpolation method for the color buffer. If Filter is not ?GL_NEAREST and Mask includes ?GL_DEPTH_BUFFER_BIT or ?GL_STENCIL_BUFFER_BIT, no data is transferred and a ?GL_INVALID_OPERATION error is generated.

If Filter is ?GL_LINEAR and the source rectangle would require sampling outside the bounds of the source framebuffer, values are read as if the ?GL_CLAMP_TO_EDGE texture wrapping mode were applied.

When the color buffer is transferred, values are taken from the read buffer of the read framebuffer and written to each of the draw buffers of the draw framebuffer.

If the source and destination rectangles overlap or are the same, and the read and draw buffers are the same, the result of the operation is undefined.

See **external** documentation.

renderbufferStorageMultisample(Target, Samples, Internalformat, Width, Height) -> ok

Types:

```
Target = enum()
Samples = integer()
Internalformat = enum()
Width = integer()
Height = integer()
```

Establish data storage, format, dimensions and sample count of a renderbuffer object's image

gl:renderbufferStorageMultisample establishes the data storage, format, dimensions and number of samples of a renderbuffer object's image.

The target of the operation, specified by Target must be ?GL_RENDERBUFFER. Internalformat specifies the internal format to be used for the renderbuffer object's storage and must be a color-renderable, depth-renderable, or stencil-renderable format. Width and Height are the dimensions, in pixels, of the renderbuffer. Both Width and Height must be less than or equal to the value of ?GL_MAX_RENDERBUFFER_SIZE . Samples specifies the number of samples to be used for the renderbuffer object's image, and must be less than or equal to the value of ?GL_MAX_SAMPLES. If Internalformat is a signed or unsigned integer format then Samples must be less than or equal to the value of ?GL_MAX_INTEGER_SAMPLES.

Upon success, gl:renderbufferStorageMultisample deletes any existing data store for the renderbuffer image and the contents of the data store after calling gl:renderbufferStorageMultisample are undefined.

See **external** documentation.

framebufferTextureLayer(Target, Attachment, Texture, Level, Layer) -> ok

Types:

Target = enum()
Attachment = enum()
Texture = integer()
Level = integer()
Layer = integer()

See *framebufferTexture/4*

framebufferTextureFaceARB(Target, Attachment, Texture, Level, Face) -> ok

Types:

Target = enum()
Attachment = enum()
Texture = integer()
Level = integer()
Face = enum()

See *framebufferTexture/4*

flushMappedBufferRange(Target, Offset, Length) -> ok

Types:

Target = enum()
Offset = integer()
Length = integer()

Indicate modifications to a range of a mapped buffer

`gl:flushMappedBufferRange` indicates that modifications have been made to a range of a mapped buffer. The buffer must previously have been mapped with the `?GL_MAP_FLUSH_EXPLICIT` flag. `Offset` and `Length` indicate the modified subrange of the mapping, in basic units. The specified subrange to flush is relative to the start of the currently mapped range of the buffer. `gl:flushMappedBufferRange` may be called multiple times to indicate distinct subranges of the mapping which require flushing.

See **external** documentation.

bindVertexArray(Array) -> ok

Types:

Array = integer()

Bind a vertex array object

`gl:bindVertexArray` binds the vertex array object with name `Array`. `Array` is the name of a vertex array object previously returned from a call to *gl:genVertexArrays/1* , or zero to break the existing vertex array object binding.

If no vertex array object with name `Array` exists, one is created when `Array` is first bound. If the bind is successful no change is made to the state of the vertex array object, and any previous vertex array object binding is broken.

See **external** documentation.

deleteVertexArrays(Arrays) -> ok

Types:

Arrays = [integer()]

Delete vertex array objects

`gl:deleteVertexArrays` deletes `N` vertex array objects whose names are stored in the array addressed by `Arrays` . Once a vertex array object is deleted it has no contents and its name is again unused. If a vertex array object that is currently bound is deleted, the binding for that object reverts to zero and the default vertex array becomes current. Unused names in `Arrays` are silently ignored, as is the value zero.

See **external** documentation.

genVertexArrays(N) -> [integer()]

Types:

N = integer()

Generate vertex array object names

`gl:genVertexArrays` returns `N` vertex array object names in `Arrays` . There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to `gl:genVertexArrays` .

Vertex array object names returned by a call to `gl:genVertexArrays` are not returned by subsequent calls, unless they are first deleted with *gl:deleteVertexArrays/1* .

The names returned in `Arrays` are marked as used, for the purposes of `gl:genVertexArrays` only, but they acquire state and type only when they are first bound.

See **external** documentation.

isVertexArray(Array) -> 0 | 1

Types:

Array = integer()

Determine if a name corresponds to a vertex array object

`gl:isVertexArray` returns `?GL_TRUE` if `Array` is currently the name of a renderbuffer object. If `Renderbuffer` is zero, or if `Array` is not the name of a renderbuffer object, or if an error occurs, `gl:isVertexArray` returns `?GL_FALSE` . If `Array` is a name returned by *gl:genVertexArrays/1* , by that has not yet been bound through a call to *gl:bindVertexArray/1* , then the name is not a vertex array object and `gl:isVertexArray` returns `?GL_FALSE`.

See **external** documentation.

getUniformIndices(Program, UniformNames) -> [integer()]

Types:

Program = integer()

UniformNames = [string()]

Retrieve the index of a named uniform block

`gl:getUniformIndices` retrieves the indices of a number of uniforms within `Program` .

`Program` must be the name of a program object for which the command *gl:linkProgram/1* must have been called in the past, although it is not required that *gl:linkProgram/1* must have succeeded. The link could have failed because the number of active uniforms exceeded the limit.

`UniformCount` indicates both the number of elements in the array of names `UniformNames` and the number of indices that may be written to `UniformIndices` .

UniformNames contains a list of UniformCount name strings identifying the uniform names to be queried for indices. For each name string in UniformNames , the index assigned to the active uniform of that name will be written to the corresponding element of UniformIndices . If a string in UniformNames is not the name of an active uniform, the special value ?GL_INVALID_INDEX will be written to the corresponding element of UniformIndices .

If an error occurs, nothing is written to UniformIndices .

See **external** documentation.

getActiveUniformsiv(Program, UniformIndices, Pname) -> [integer()]

Types:

Program = integer()
UniformIndices = [integer()]
Pname = enum()

glGetActiveUniforms

See **external** documentation.

getActiveUniformName(Program, UniformIndex, BufSize) -> string()

Types:

Program = integer()
UniformIndex = integer()
BufSize = integer()

Query the name of an active uniform

gl:getActiveUniformName returns the name of the active uniform at UniformIndex within Program . If UniformName is not NULL, up to BufSize characters (including a nul-terminator) will be written into the array whose address is specified by UniformName . If Length is not NULL, the number of characters that were (or would have been) written into UniformName (not including the nul-terminator) will be placed in the variable whose address is specified in Length . If Length is NULL, no length is returned. The length of the longest uniform name in Program is given by the value of ?GL_ACTIVE_UNIFORM_MAX_LENGTH, which can be queried with *gl:getProgramiv/2* .

If gl:getActiveUniformName is not successful, nothing is written to Length or UniformName .

Program must be the name of a program for which the command *gl:linkProgram/1* has been issued in the past. It is not necessary for Program to have been linked successfully. The link could have failed because the number of active uniforms exceeded the limit.

UniformIndex must be an active uniform index of the program Program , in the range zero to ? GL_ACTIVE_UNIFORMS - 1. The value of ?GL_ACTIVE_UNIFORMS can be queried with *gl:getProgramiv/2* .

See **external** documentation.

getUniformBlockIndex(Program, UniformBlockName) -> integer()

Types:

Program = integer()
UniformBlockName = string()

Retrieve the index of a named uniform block

gl:getUniformBlockIndex retrieves the index of a uniform block within Program .

Program must be the name of a program object for which the command *gl:linkProgram/1* must have been called in the past, although it is not required that *gl:linkProgram/1* must have succeeded. The link could have failed because the number of active uniforms exceeded the limit.

UniformBlockName must contain a nul-terminated string specifying the name of the uniform block.

gl:getUniformBlockIndex returns the uniform block index for the uniform block named UniformBlockName of Program . If UniformBlockName does not identify an active uniform block of Program , gl:getUniformBlockIndex returns the special identifier, ?GL_INVALID_INDEX. Indices of the active uniform blocks of a program are assigned in consecutive order, beginning with zero.

See **external** documentation.

getActiveUniformBlockiv(Program, UniformBlockIndex, Pname, Params) -> ok

Types:

```
Program = integer()
UniformBlockIndex = integer()
Pname = enum()
Params = mem()
```

Query information about an active uniform block

gl:getActiveUniformBlockiv retrieves information about an active uniform block within Program .

Program must be the name of a program object for which the command *gl:linkProgram/1* must have been called in the past, although it is not required that *gl:linkProgram/1* must have succeeded. The link could have failed because the number of active uniforms exceeded the limit.

UniformBlockIndex is an active uniform block index of Program , and must be less than the value of ?GL_ACTIVE_UNIFORM_BLOCKS.

Upon success, the uniform block parameter(s) specified by Pname are returned in Params . If an error occurs, nothing will be written to Params .

If Pname is ?GL_UNIFORM_BLOCK_BINDING, then the index of the uniform buffer binding point last selected by the uniform block specified by UniformBlockIndex for Program is returned. If no uniform block has been previously specified, zero is returned.

If Pname is ?GL_UNIFORM_BLOCK_DATA_SIZE, then the implementation-dependent minimum total buffer object size, in basic machine units, required to hold all active uniforms in the uniform block identified by UniformBlockIndex is returned. It is neither guaranteed nor expected that a given implementation will arrange uniform values as tightly packed in a buffer object. The exception to this is the std140 uniform block layout , which guarantees specific packing behavior and does not require the application to query for offsets and strides. In this case the minimum size may still be queried, even though it is determined in advance based only on the uniform block declaration.

If Pname is ?GL_UNIFORM_BLOCK_NAME_LENGTH, then the total length (including the nul terminator) of the name of the uniform block identified by UniformBlockIndex is returned.

If Pname is ?GL_UNIFORM_BLOCK_ACTIVE_UNIFORMS, then the number of active uniforms in the uniform block identified by UniformBlockIndex is returned.

If Pname is ?GL_UNIFORM_BLOCK_ACTIVE_UNIFORM_INDICES, then a list of the active uniform indices for the uniform block identified by UniformBlockIndex is returned. The number of elements that will be written to Params is the value of ?GL_UNIFORM_BLOCK_ACTIVE_UNIFORMS for UniformBlockIndex .

If Pname is ?GL_UNIFORM_BLOCK_REFERENCED_BY_VERTEX_SHADER, ?GL_UNIFORM_BLOCK_REFERENCED_BY_GEOMETRY_SHADER , or ?GL_UNIFORM_BLOCK_REFERENCED_BY_FRAGMENT_SHADER, then a boolean value indicating whether the

uniform block identified by UniformBlockIndex is referenced by the vertex, geometry, or fragment programming stages of program, respectively, is returned.

See **external** documentation.

getActiveUniformBlockName(Program, UniformBlockIndex, BufSize) -> string()

Types:

Program = integer()
UniformBlockIndex = integer()
BufSize = integer()

Retrieve the name of an active uniform block

gl:getActiveUniformBlockName retrieves the name of the active uniform block at UniformBlockIndex within Program .

Program must be the name of a program object for which the command *gl:linkProgram/1* must have been called in the past, although it is not required that *gl:linkProgram/1* must have succeeded. The link could have failed because the number of active uniforms exceeded the limit.

UniformBlockIndex is an active uniform block index of Program , and must be less than the value of ? GL_ACTIVE_UNIFORM_BLOCKS.

Upon success, the name of the uniform block identified by UnifomBlockIndex is returned into UniformBlockName . The name is nul-terminated. The actual number of characters written into UniformBlockName , excluding the nul terminator, is returned in Length . If Length is NULL, no length is returned.

BufSize contains the maximum number of characters (including the nul terminator) that will be written into UniformBlockName .

If an error occurs, nothing will be written to UniformBlockName or Length .

See **external** documentation.

uniformBlockBinding(Program, UniformBlockIndex, UniformBlockBinding) -> ok

Types:

Program = integer()
UniformBlockIndex = integer()
UniformBlockBinding = integer()

Assign a binding point to an active uniform block

Binding points for active uniform blocks are assigned using gl:uniformBlockBinding. Each of a program's active uniform blocks has a corresponding uniform buffer binding point. Program is the name of a program object for which the command *gl:linkProgram/1* has been issued in the past.

If successful, gl:uniformBlockBinding specifies that Program will use the data store of the buffer object bound to the binding point UniformBlockBinding to extract the values of the uniforms in the uniform block identified by UniformBlockIndex .

When a program object is linked or re-linked, the uniform buffer object binding point assigned to each of its active uniform blocks is reset to zero.

See **external** documentation.

copyBufferSubData(ReadTarget, WriteTarget, ReadOffset, WriteOffset, Size) -> ok

Types:

ReadTarget = enum()
WriteTarget = enum()
ReadOffset = integer()
WriteOffset = integer()
Size = integer()

Copy part of the data store of a buffer object to the data store of another buffer object

`gl:copyBufferSubData` copies part of the data store attached to `Readtarget` to the data store attached to `Writetarget` . The number of basic machine units indicated by `Size` is copied from the source, at offset `Readoffset` to the destination at `Writeoffset` , also in basic machine units.

`Readtarget` and `Writetarget` must be `?GL_ARRAY_BUFFER`, `?GL_COPY_READ_BUFFER` , `?GL_COPY_WRITE_BUFFER`, `?GL_ELEMENT_ARRAY_BUFFER`, `?GL_PIXEL_PACK_BUFFER` , `?GL_PIXEL_UNPACK_BUFFER`, `?GL_TEXTURE_BUFFER`, `?GL_TRANSFORM_FEEDBACK_BUFFER` or `?GL_UNIFORM_BUFFER`. Any of these targets may be used, although the targets `?GL_COPY_READ_BUFFER` and `?GL_COPY_WRITE_BUFFER` are provided specifically to allow copies between buffers without disturbing other GL state.

`Readoffset` , `Writeoffset` and `Size` must all be greater than or equal to zero. Furthermore, `Readoffset` + `Size` must not exceeed the size of the buffer object bound to `Readtarget` , and `Readoffset` + `Size` must not exceeed the size of the buffer bound to `Writetarget` . If the same buffer object is bound to both `Readtarget` and `Writetarget` , then the ranges specified by `Readoffset` , `Writeoffset` and `Size` must not overlap.

See **external** documentation.

drawElementsBaseVertex(Mode, Count, Type, Indices, Basevertex) -> ok

Types:

Mode = enum()
Count = integer()
Type = enum()
Indices = offset() | mem()
Basevertex = integer()

Render primitives from array data with a per-element offset

`gl:drawElementsBaseVertex` behaves identically to *gl:drawElements/4* except that the `i`th element transferred by the corresponding draw call will be taken from element `Indices` [i] + `Basevertex` of each enabled array. If the resulting value is larger than the maximum value representable by `Type` , it is as if the calculation were upconverted to 32-bit unsigned integers (with wrapping on overflow conditions). The operation is undefined if the sum would be negative.

See **external** documentation.

drawRangeElementsBaseVertex(Mode, Start, End, Count, Type, Indices, Basevertex) -> ok

Types:

Mode = enum()
Start = integer()
End = integer()

Count = integer()
Type = enum()
Indices = offset() | mem()
Basevertex = integer()

Render primitives from array data with a per-element offset

`gl:drawRangeElementsBaseVertex` is a restricted form of *gl:drawElementsBaseVertex/5* . `Mode` , `Start` , `End` , `Count` and `Basevertex` match the corresponding arguments to *gl:drawElementsBaseVertex/5* , with the additional constraint that all values in the array `Indices` must lie between `Start` and `End` , inclusive, prior to adding `Basevertex` . Index values lying outside the range [`Start` , `End`] are treated in the same way as *gl:drawElementsBaseVertex/5* . The `i` th element transferred by the corresponding draw call will be taken from element `Indices` [i] + `Basevertex` of each enabled array. If the resulting value is larger than the maximum value representable by `Type` , it is as if the calculation were upconverted to 32-bit unsigned integers (with wrapping on overflow conditions). The operation is undefined if the sum would be negative.

See **external** documentation.

drawElementsInstancedBaseVertex(Mode, Count, Type, Indices, Primcount, Basevertex) -> ok

Types:

Mode = enum()
Count = integer()
Type = enum()
Indices = offset() | mem()
Primcount = integer()
Basevertex = integer()

Render multiple instances of a set of primitives from array data with a per-element offset

`gl:drawElementsInstancedBaseVertex` behaves identically to *gl:drawElementsInstanced/5* except that the `i`th element transferred by the corresponding draw call will be taken from element `Indices` [i] + `Basevertex` of each enabled array. If the resulting value is larger than the maximum value representable by `Type` , it is as if the calculation were upconverted to 32-bit unsigned integers (with wrapping on overflow conditions). The operation is undefined if the sum would be negative.

See **external** documentation.

provokingVertex(Mode) -> ok

Types:

Mode = enum()

Specifiy the vertex to be used as the source of data for flat shaded varyings

`Flatshading` a vertex shader varying output means to assign all vetices of the primitive the same value for that output. The vertex from which these values is derived is known as the `provoking vertex` and `gl:provokingVertex` specifies which vertex is to be used as the source of data for flat shaded varyings.

`ProvokeMode` must be either `?GL_FIRST_VERTEX_CONVENTION` or `?GL_LAST_VERTEX_CONVENTION` , and controls the selection of the vertex whose values are assigned to flatshaded varying outputs. The interpretation of these values for the supported primitive types is:

`Primitive Type of PolygoniFirst Vertex Convention Last Vertex Convention`
point `i i`
independent line `2i - 1 2i`

line loop i

i + 1, if i < n

1, if i = n
line strip ii + 1
independent triangle 3i - 2 3i
triangle strip ii + 2
triangle fan i + 1 i + 2
line adjacency 4i - 2 4i - 1
line strip adjacency i + 1 i + 2
triangle adjacency 6i - 5 6i - 1
triangle strip adjacency 2i - 1 2i + 3

If a vertex or geometry shader is active, user-defined varying outputs may be flatshaded by using the flat qualifier when declaring the output.

See **external** documentation.

fenceSync(Condition, Flags) -> integer()

Types:

Condition = enum()
Flags = integer()

Create a new sync object and insert it into the GL command stream

gl:fenceSync creates a new fence sync object, inserts a fence command into the GL command stream and associates it with that sync object, and returns a non-zero name corresponding to the sync object.

When the specified Condition of the sync object is satisfied by the fence command, the sync object is signaled by the GL, causing any *gl:waitSync/3* , *gl:clientWaitSync/3* commands blocking in Sync to unblock. No other state is affected by gl:fenceSync or by the execution of the associated fence command.

Condition must be ?GL_SYNC_GPU_COMMANDS_COMPLETE. This condition is satisfied by completion of the fence command corresponding to the sync object and all preceding commands in the same command stream. The sync object will not be signaled until all effects from these commands on GL client and server state and the framebuffer are fully realized. Note that completion of the fence command occurs once the state of the corresponding sync object has been changed, but commands waiting on that sync object may not be unblocked until after the fence command completes.

See **external** documentation.

isSync(Sync) -> 0 | 1

Types:

Sync = integer()

Determine if a name corresponds to a sync object

gl:isSync returns ?GL_TRUE if Sync is currently the name of a sync object. If Sync is not the name of a sync object, or if an error occurs, gl:isSync returns ?GL_FALSE. Note that zero is not the name of a sync object.

See **external** documentation.

deleteSync(Sync) -> ok

Types:

Sync = integer()

Delete a sync object

`gl:deleteSync` deletes the sync object specified by `Sync` . If the fence command corresponding to the specified sync object has completed, or if no *gl:waitSync/3* or *gl:clientWaitSync/3* commands are blocking on `Sync` , the object is deleted immediately. Otherwise, `Sync` is flagged for deletion and will be deleted when it is no longer associated with any fence command and is no longer blocking any *gl:waitSync/3* or *gl:clientWaitSync/3* command. In either case, after `gl:deleteSync` returns, the name `Sync` is invalid and can no longer be used to refer to the sync object.

`gl:deleteSync` will silently ignore a `Sync` value of zero.

See **external** documentation.

`clientWaitSync(Sync, Flags, Timeout) -> enum()`

Types:

```
Sync = integer()
Flags = integer()
Timeout = integer()
```

Block and wait for a sync object to become signaled

`gl:clientWaitSync` causes the client to block and wait for the sync object specified by `Sync` to become signaled. If `Sync` is signaled when `gl:clientWaitSync` is called, `gl:clientWaitSync` returns immediately, otherwise it will block and wait for up to `Timeout` nanoseconds for `Sync` to become signaled.

The return value is one of four status values:

`?GL_ALREADY_SIGNALED` indicates that `Sync` was signaled at the time that `gl:clientWaitSync` was called.

`?GL_TIMEOUT_EXPIRED` indicates that at least `Timeout` nanoseconds passed and `Sync` did not become signaled.

`?GL_CONDITION_SATISFIED` indicates that `Sync` was signaled before the timeout expired.

`?GL_WAIT_FAILED` indicates that an error occurred. Additionally, an OpenGL error will be generated.

See **external** documentation.

`waitSync(Sync, Flags, Timeout) -> ok`

Types:

```
Sync = integer()
Flags = integer()
Timeout = integer()
```

Instruct the GL server to block until the specified sync object becomes signaled

`gl:waitSync` causes the GL server to block and wait until `Sync` becomes signaled. `Sync` is the name of an existing sync object upon which to wait. `Flags` and `Timeout` are currently not used and must be set to zero and the special value `?GL_TIMEOUT_IGNORED` , respectively

`Flags` and `Timeout` are placeholders for anticipated future extensions of sync object capabilities. They must have these reserved values in order that existing code calling `gl:waitSync` operate properly in the presence of such extensions.. `gl:waitSync` will always wait no longer than an implementation-dependent timeout. The duration of this timeout in nanoseconds may be queried by calling *gl:getBooleanv/1* with the parameter `?GL_MAX_SERVER_WAIT_TIMEOUT` . There is currently no way to determine whether `gl:waitSync` unblocked because the timeout expired or because the sync object being waited on was signaled.

If an error occurs, `gl:waitSync` does not cause the GL server to block.

See **external** documentation.

getInteger64v(Pname) -> [integer()]

Types:

Pname = enum()

See *getBooleanv/1*

getSynciv(Sync, Pname, BufSize) -> [integer()]

Types:

Sync = integer()
Pname = enum()
BufSize = integer()

Query the properties of a sync object

`gl:getSynciv` retrieves properties of a sync object. `Sync` specifies the name of the sync object whose properties to retrieve.

On success, `gl:getSynciv` replaces up to `BufSize` integers in `Values` with the corresponding property values of the object being queried. The actual number of integers replaced is returned in the variable whose address is specified in `Length` . If `Length` is NULL, no length is returned.

If `Pname` is `?GL_OBJECT_TYPE`, a single value representing the specific type of the sync object is placed in `Values` . The only type supported is `?GL_SYNC_FENCE` .

If `Pname` is `?GL_SYNC_STATUS`, a single value representing the status of the sync object (`?GL_SIGNALED` or `?GL_UNSIGNALED`) is placed in `Values` .

If `Pname` is `?GL_SYNC_CONDITION`, a single value representing the condition of the sync object is placed in `Values` . The only condition supported is `?GL_SYNC_GPU_COMMANDS_COMPLETE` .

If `Pname` is `?GL_SYNC_FLAGS`, a single value representing the flags with which the sync object was created is placed in `Values` . No flags are currently supported

`Flags` is expected to be used in future extensions to the sync objects..

If an error occurs, nothing will be written to `Values` or `Length` .

See **external** documentation.

texImage2DMultisample(Target, Samples, Internalformat, Width, Height, Fixedsamplelocations) -> ok

Types:

Target = enum()
Samples = integer()
Internalformat = integer()
Width = integer()
Height = integer()
Fixedsamplelocations = 0 | 1

Establish the data storage, format, dimensions, and number of samples of a multisample texture's image

`gl:texImage2DMultisample` establishes the data storage, format, dimensions and number of samples of a multisample texture's image.

`Target` must be `?GL_TEXTURE_2D_MULTISAMPLE` or `?GL_PROXY_TEXTURE_2D_MULTISAMPLE` . `Width` and `Height` are the dimensions in texels of the texture, and must be in the range zero to `?GL_MAX_TEXTURE_SIZE` - 1. `Samples` specifies the number of samples in the image and must be in the range zero to `?GL_MAX_SAMPLES` - 1.

Internalformat must be a color-renderable, depth-renderable, or stencil-renderable format.

If Fixedsamplelocations is ?GL_TRUE, the image will use identical sample locations and the same number of samples for all texels in the image, and the sample locations will not depend on the internal format or size of the image.

When a multisample texture is accessed in a shader, the access takes one vector of integers describing which texel to fetch and an integer corresponding to the sample numbers describing which sample within the texel to fetch. No standard sampling instructions are allowed on the multisample texture targets.

See **external** documentation.

texImage3DMultisample(Target, Samples, Internalformat, Width, Height, Depth, Fixedsamplelocations) -> ok

Types:

Target = enum()
Samples = integer()
Internalformat = integer()
Width = integer()
Height = integer()
Depth = integer()
Fixedsamplelocations = 0 | 1

Establish the data storage, format, dimensions, and number of samples of a multisample texture's image

gl:texImage3DMultisample establishes the data storage, format, dimensions and number of samples of a multisample texture's image.

Target must be ?GL_TEXTURE_2D_MULTISAMPLE_ARRAY or ?GL_PROXY_TEXTURE_2D_MULTISAMPLE_ARRAY . Width and Height are the dimensions in texels of the texture, and must be in the range zero to ?GL_MAX_TEXTURE_SIZE - 1. Depth is the number of array slices in the array texture's image. Samples specifies the number of samples in the image and must be in the range zero to ?GL_MAX_SAMPLES - 1.

Internalformat must be a color-renderable, depth-renderable, or stencil-renderable format.

If Fixedsamplelocations is ?GL_TRUE, the image will use identical sample locations and the same number of samples for all texels in the image, and the sample locations will not depend on the internal format or size of the image.

When a multisample texture is accessed in a shader, the access takes one vector of integers describing which texel to fetch and an integer corresponding to the sample numbers describing which sample within the texel to fetch. No standard sampling instructions are allowed on the multisample texture targets.

See **external** documentation.

getMultisamplefv(Pname, Index) -> {float(), float()}

Types:

Pname = enum()
Index = integer()

Retrieve the location of a sample

gl:getMultisamplefv queries the location of a given sample. Pname specifies the sample parameter to retrieve and must be ?GL_SAMPLE_POSITION. Index corresponds to the sample for which the location should be returned. The sample location is returned as two floating-point values in Val[0] and Val[1] , each between 0 and 1, corresponding to the X and Y locations respectively in the GL pixel space of that sample. (0.5, 0.5) this corresponds to the pixel center. Index must be between zero and the value of ?GL_SAMPLES - 1.

If the multisample mode does not have fixed sample locations, the returned values may only reflect the locations of samples within some pixels.

See **external** documentation.

sampleMaski(Index, Mask) -> ok

Types:

Index = integer()
Mask = integer()

Set the value of a sub-word of the sample mask

`gl:sampleMaski` sets one 32-bit sub-word of the multi-word sample mask, `?GL_SAMPLE_MASK_VALUE` .

`MaskIndex` specifies which 32-bit sub-word of the sample mask to update, and `Mask` specifies the new value to use for that sub-word. `MaskIndex` must be less than the value of `?GL_MAX_SAMPLE_MASK_WORDS`. Bit `B` of mask word `M` corresponds to sample 32 x `M` + `B`.

See **external** documentation.

namedStringARB(Type, Name, String) -> ok

Types:

Type = enum()
Name = string()
String = string()

glNamedStringARB

See **external** documentation.

deleteNamedStringARB(Name) -> ok

Types:

Name = string()

glDeleteNamedStringARB

See **external** documentation.

compileShaderIncludeARB(Shader, Path) -> ok

Types:

Shader = integer()
Path = [string()]

glCompileShaderIncludeARB

See **external** documentation.

isNamedStringARB(Name) -> 0 | 1

Types:

Name = string()

glIsNamedStringARB

See **external** documentation.

getNamedStringARB(Name, BufSize) -> string()

Types:

Name = string()

BufSize = integer()

glGetNamedStringARB

See **external** documentation.

getNamedStringivARB(Name, Pname) -> integer()

Types:

Name = string()

Pname = enum()

glGetNamedStringARB

See **external** documentation.

bindFragDataLocationIndexed(Program, ColorNumber, Index, Name) -> ok

Types:

Program = integer()

ColorNumber = integer()

Index = integer()

Name = string()

glBindFragDataLocationIndexe

See **external** documentation.

getFragDataIndex(Program, Name) -> integer()

Types:

Program = integer()

Name = string()

Query the bindings of color indices to user-defined varying out variables

`gl:getFragDataIndex` returns the index of the fragment color to which the variable `Name` was bound when the program object `Program` was last linked. If `Name` is not a varying out variable of `Program` , or if an error occurs, -1 will be returned.

See **external** documentation.

genSamplers(Count) -> [integer()]

Types:

Count = integer()

Generate sampler object names

`gl:genSamplers` returns `N` sampler object names in `Samplers` . There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to `gl:genSamplers` .

Sampler object names returned by a call to `gl:genSamplers` are not returned by subsequent calls, unless they are first deleted with *gl:deleteSamplers/1* .

The names returned in `Samplers` are marked as used, for the purposes of `gl:genSamplers` only, but they acquire state and type only when they are first bound.

See **external** documentation.

deleteSamplers(Samplers) -> ok

Types:

Samplers = [integer()]

Delete named sampler objects

`gl:deleteSamplers` deletes `N` sampler objects named by the elements of the array `Ids` . After a sampler object is deleted, its name is again unused. If a sampler object that is currently bound to a sampler unit is deleted, it is as though *gl:bindSampler/2* is called with unit set to the unit the sampler is bound to and sampler zero. Unused names in samplers are silently ignored, as is the reserved name zero.

See **external** documentation.

isSampler(Sampler) -> 0 | 1

Types:

Sampler = integer()

Determine if a name corresponds to a sampler object

`gl:isSampler` returns `?GL_TRUE` if `Id` is currently the name of a sampler object. If `Id` is zero, or is a non-zero value that is not currently the name of a sampler object, or if an error occurs, `gl:isSampler` returns `?GL_FALSE`.

A name returned by *gl:genSamplers/1* , is the name of a sampler object.

See **external** documentation.

bindSampler(Unit, Sampler) -> ok

Types:

Unit = integer()

Sampler = integer()

Bind a named sampler to a texturing target

`gl:bindSampler` binds `Sampler` to the texture unit at index `Unit` . `Sampler` must be zero or the name of a sampler object previously returned from a call to *gl:genSamplers/1* . `Unit` must be less than the value of `?GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS`.

When a sampler object is bound to a texture unit, its state supersedes that of the texture object bound to that texture unit. If the sampler name zero is bound to a texture unit, the currently bound texture's sampler state becomes active. A single sampler object may be bound to multiple texture units simultaneously.

See **external** documentation.

samplerParameteri(Sampler, Pname, Param) -> ok

Types:

Sampler = integer()

Pname = enum()

Param = integer()

Set sampler parameters

gl:samplerParameter assigns the value or values in Params to the sampler parameter specified as Pname . Sampler specifies the sampler object to be modified, and must be the name of a sampler object previously returned from a call to *gl:genSamplers/1* . The following symbols are accepted in Pname :

?GL_TEXTURE_MIN_FILTER: The texture minifying function is used whenever the pixel being textured maps to an area greater than one texture element. There are six defined minifying functions. Two of them use the nearest one or nearest four texture elements to compute the texture value. The other four use mipmaps.

A mipmap is an ordered set of arrays representing the same image at progressively lower resolutions. If the texture has dimensions 2 n*2 m, there are max(n m)+1 mipmaps. The first mipmap is the original texture, with dimensions 2 n*2 m. Each subsequent mipmap has dimensions 2(k-1)*2(l-1), where 2 k*2 l are the dimensions of the previous mipmap, until either k= 0 or l= 0. At that point, subsequent mipmaps have dimension 1*2(l-1) or 2(k-1)*1 until the final mipmap, which has dimension 1*1. To define the mipmaps, call *gl:texImage1D/8* , *gl:texImage2D/9* , *gl:texImage3D/10* , *gl:copyTexImage1D/7* , or *gl:copyTexImage2D/8* with the level argument indicating the order of the mipmaps. Level 0 is the original texture; level max(n m) is the final 1*1 mipmap.

Params supplies a function for minifying the texture as one of the following:

?GL_NEAREST: Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured.

?GL_LINEAR: Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of ?GL_TEXTURE_WRAP_S and ?GL_TEXTURE_WRAP_T, and on the exact mapping.

?GL_NEAREST_MIPMAP_NEAREST: Chooses the mipmap that most closely matches the size of the pixel being textured and uses the ?GL_NEAREST criterion (the texture element nearest to the center of the pixel) to produce a texture value.

?GL_LINEAR_MIPMAP_NEAREST: Chooses the mipmap that most closely matches the size of the pixel being textured and uses the ?GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value.

?GL_NEAREST_MIPMAP_LINEAR: Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the ?GL_NEAREST criterion (the texture element nearest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

?GL_LINEAR_MIPMAP_LINEAR: Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the ?GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

As more texture elements are sampled in the minification process, fewer aliasing artifacts will be apparent. While the ?GL_NEAREST and ?GL_LINEAR minification functions can be faster than the other four, they sample only one or four texture elements to determine the texture value of the pixel being rendered and can produce moire patterns or ragged transitions. The initial value of ?GL_TEXTURE_MIN_FILTER is ?GL_NEAREST_MIPMAP_LINEAR .

?GL_TEXTURE_MAG_FILTER: The texture magnification function is used when the pixel being textured maps to an area less than or equal to one texture element. It sets the texture magnification function to either ?GL_NEAREST or ?GL_LINEAR (see below). ?GL_NEAREST is generally faster than ?GL_LINEAR, but it can produce textured images with sharper edges because the transition between texture elements is not as smooth. The initial value of ?GL_TEXTURE_MAG_FILTER is ?GL_LINEAR.

?GL_NEAREST: Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured.

?GL_LINEAR: Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of ?GL_TEXTURE_WRAP_S and ?GL_TEXTURE_WRAP_T, and on the exact mapping.

?GL_TEXTURE_MIN_LOD: Sets the minimum level-of-detail parameter. This floating-point value limits the selection of highest resolution mipmap (lowest mipmap level). The initial value is -1000.

?GL_TEXTURE_MAX_LOD: Sets the maximum level-of-detail parameter. This floating-point value limits the selection of the lowest resolution mipmap (highest mipmap level). The initial value is 1000.

?GL_TEXTURE_WRAP_S: Sets the wrap parameter for texture coordinate s to either ?GL_CLAMP_TO_EDGE , ?GL_MIRRORED_REPEAT, or ?GL_REPEAT. ?GL_CLAMP_TO_BORDER causes the s coordinate to be clamped to the range [(-1 2/N) 1+(1 2/N)], where N is the size of the texture in the direction of clamping.?GL_CLAMP_TO_EDGE causes s coordinates to be clamped to the range [(1 2/N) 1-(1 2/N)], where N is the size of the texture in the direction of clamping. ?GL_REPEAT causes the integer part of the s coordinate to be ignored; the GL uses only the fractional part, thereby creating a repeating pattern. ?GL_MIRRORED_REPEAT causes the s coordinate to be set to the fractional part of the texture coordinate if the integer part of s is even; if the integer part of s is odd, then the s texture coordinate is set to 1-frac(s), where frac(s) represents the fractional part of s. Initially, ?GL_TEXTURE_WRAP_S is set to ?GL_REPEAT.

?GL_TEXTURE_WRAP_T: Sets the wrap parameter for texture coordinate t to either ?GL_CLAMP_TO_EDGE , ?GL_MIRRORED_REPEAT, or ?GL_REPEAT. See the discussion under ?GL_TEXTURE_WRAP_S . Initially, ?GL_TEXTURE_WRAP_T is set to ?GL_REPEAT.

?GL_TEXTURE_WRAP_R: Sets the wrap parameter for texture coordinate r to either ?GL_CLAMP_TO_EDGE , ?GL_MIRRORED_REPEAT, or ?GL_REPEAT. See the discussion under ?GL_TEXTURE_WRAP_S . Initially, ?GL_TEXTURE_WRAP_R is set to ?GL_REPEAT.

?GL_TEXTURE_BORDER_COLOR: The data in Params specifies four values that define the border values that should be used for border texels. If a texel is sampled from the border of the texture, the values of ?GL_TEXTURE_BORDER_COLOR are interpreted as an RGBA color to match the texture's internal format and substituted for the non-existent texel data. If the texture contains depth components, the first component of ?GL_TEXTURE_BORDER_COLOR is interpreted as a depth value. The initial value is (0.0, 0.0, 0.0, 0.0).

?GL_TEXTURE_COMPARE_MODE: Specifies the texture comparison mode for currently bound textures. That is, a texture whose internal format is ?GL_DEPTH_COMPONENT_*; see *gl:texImage2D/9*) Permissible values are:

?GL_COMPARE_REF_TO_TEXTURE: Specifies that the interpolated and clamped r texture coordinate should be compared to the value in the currently bound texture. See the discussion of ?GL_TEXTURE_COMPARE_FUNC for details of how the comparison is evaluated. The result of the comparison is assigned to the red channel.

?GL_NONE: Specifies that the red channel should be assigned the appropriate value from the currently bound texture.

?GL_TEXTURE_COMPARE_FUNC: Specifies the comparison operator used when ?GL_TEXTURE_COMPARE_MODE is set to ?GL_COMPARE_REF_TO_TEXTURE. Permissible values are:

Texture Comparison FunctionComputed result
?GL_LEQUAL result={1.0 0.0   r<=(D t) r>(D t))
?GL_GEQUAL result={1.0 0.0   r>=(D t) r<(D t))
?GL_LESS result={1.0 0.0   r< (D t) r>=(D t))
?GL_GREATER result={1.0 0.0   r>(D t) r<=(D t))
?GL_EQUAL result={1.0 0.0   r=(D t) r≠(D t))
?GL_NOTEQUAL result={1.0 0.0   r≠(D t) r=(D t))
?GL_ALWAYS result= 1.0
?GL_NEVER result= 0.0

where r is the current interpolated texture coordinate, and D t is the texture value sampled from the currently bound texture. result is assigned to R t.

See **external** documentation.

samplerParameteriv(Sampler, Pname, Param) -> ok

Types:

```
Sampler = integer()
Pname = enum()
Param = [integer()]
```

See *samplerParameteri/3*

samplerParameterf(Sampler, Pname, Param) -> ok

Types:

```
Sampler = integer()
Pname = enum()
Param = float()
```

See *samplerParameteri/3*

samplerParameterfv(Sampler, Pname, Param) -> ok

Types:

```
Sampler = integer()
Pname = enum()
Param = [float()]
```

See *samplerParameteri/3*

samplerParameterIiv(Sampler, Pname, Param) -> ok

Types:

```
Sampler = integer()
Pname = enum()
Param = [integer()]
```

See *samplerParameteri/3*

samplerParameterIuiv(Sampler, Pname, Param) -> ok

Types:

```
Sampler = integer()
Pname = enum()
Param = [integer()]
```

glSamplerParameterI

See **external** documentation.

getSamplerParameteriv(Sampler, Pname) -> [integer()]

Types:

```
Sampler = integer()
Pname = enum()
```

Return sampler parameter values

`gl:getSamplerParameter` returns in `Params` the value or values of the sampler parameter specified as `Pname` . `Sampler` defines the target sampler, and must be the name of an existing sampler object, returned from a previous call to *gl:genSamplers/1* . `Pname` accepts the same symbols as *gl:samplerParameteri/3* , with the same interpretations:

?GL_TEXTURE_MAG_FILTER: Returns the single-valued texture magnification filter, a symbolic constant. The initial value is ?GL_LINEAR.

?GL_TEXTURE_MIN_FILTER: Returns the single-valued texture minification filter, a symbolic constant. The initial value is ?GL_NEAREST_MIPMAP_LINEAR.

?GL_TEXTURE_MIN_LOD: Returns the single-valued texture minimum level-of-detail value. The initial value is -1000.

?GL_TEXTURE_MAX_LOD: Returns the single-valued texture maximum level-of-detail value. The initial value is 1000.

?GL_TEXTURE_WRAP_S: Returns the single-valued wrapping function for texture coordinate s, a symbolic constant. The initial value is ?GL_REPEAT.

?GL_TEXTURE_WRAP_T: Returns the single-valued wrapping function for texture coordinate t, a symbolic constant. The initial value is ?GL_REPEAT.

?GL_TEXTURE_WRAP_R: Returns the single-valued wrapping function for texture coordinate r, a symbolic constant. The initial value is ?GL_REPEAT.

?GL_TEXTURE_BORDER_COLOR: Returns four integer or floating-point numbers that comprise the RGBA color of the texture border. Floating-point values are returned in the range [0 1]. Integer values are returned as a linear mapping of the internal floating-point representation such that 1.0 maps to the most positive representable integer and -1.0 maps to the most negative representable integer. The initial value is (0, 0, 0, 0).

?GL_TEXTURE_COMPARE_MODE: Returns a single-valued texture comparison mode, a symbolic constant. The initial value is ?GL_NONE. See *gl:samplerParameteri/3* .

?GL_TEXTURE_COMPARE_FUNC: Returns a single-valued texture comparison function, a symbolic constant. The initial value is ?GL_LEQUAL. See *gl:samplerParameteri/3* .

See **external** documentation.

`getSamplerParameterIiv(Sampler, Pname) -> [integer()]`

Types:

```
Sampler = integer()
Pname = enum()
```

See *getSamplerParameteriv/2*

`getSamplerParameterfv(Sampler, Pname) -> [float()]`

Types:

```
Sampler = integer()
Pname = enum()
```

See *getSamplerParameteriv/2*

`getSamplerParameterIuiv(Sampler, Pname) -> [integer()]`

Types:

```
Sampler = integer()
Pname = enum()
```

glGetSamplerParameterI

See **external** documentation.

queryCounter(Id, Target) -> ok

Types:

Id = integer()

Target = enum()

Record the GL time into a query object after all previous commands have reached the GL server but have not yet necessarily executed.

`gl:queryCounter` causes the GL to record the current time into the query object named `Id` . `Target` must be `?GL_TIMESTAMP`. The time is recorded after all previous commands on the GL client and server state and the framebuffer have been fully realized. When the time is recorded, the query result for that object is marked available. `gl:queryCounter` timer queries can be used within a *gl:beginQuery/2* / *gl:beginQuery/2* block where the target is `?GL_TIME_ELAPSED` and it does not affect the result of that query object.

See **external** documentation.

getQueryObjecti64v(Id, Pname) -> integer()

Types:

Id = integer()

Pname = enum()

glGetQueryObjecti64v

See **external** documentation.

getQueryObjectui64v(Id, Pname) -> integer()

Types:

Id = integer()

Pname = enum()

glGetQueryObjectui64v

See **external** documentation.

drawArraysIndirect(Mode, Indirect) -> ok

Types:

Mode = enum()

Indirect = offset() | mem()

Render primitives from array data, taking parameters from memory

`gl:drawArraysIndirect` specifies multiple geometric primitives with very few subroutine calls. `gl:drawArraysIndirect` behaves similarly to *gl:drawArraysInstancedBaseInstance/5* , execept that the parameters to *gl:drawArraysInstancedBaseInstance/5* are stored in memory at the address given by `Indirect` .

The parameters addressed by `Indirect` are packed into a structure that takes the form (in C): typedef struct { uint count; uint primCount; uint first; uint baseInstance; } DrawArraysIndirectCommand; const DrawArraysIndirectCommand *cmd = (const DrawArraysIndirectCommand *)indirect; glDrawArraysInstancedBaseInstance(mode, cmd->first, cmd->count, cmd->primCount, cmd->baseInstance);

If a buffer is bound to the `?GL_DRAW_INDIRECT_BUFFER` binding at the time of a call to `gl:drawArraysIndirect`, `Indirect` is interpreted as an offset, in basic machine units, into that buffer and the parameter data is read from the buffer rather than from client memory.

In contrast to *gl:drawArraysInstancedBaseInstance/5* , the first member of the parameter structure is unsigned, and out-of-range indices do not generate an error.

Vertex attributes that are modified by `gl:drawArraysIndirect` have an unspecified value after `gl:drawArraysIndirect` returns. Attributes that aren't modified remain well defined.

See **external** documentation.

`drawElementsIndirect(Mode, Type, Indirect) -> ok`

Types:

```
Mode = enum()
Type = enum()
Indirect = offset() | mem()
```

Render indexed primitives from array data, taking parameters from memory

`gl:drawElementsIndirect` specifies multiple indexed geometric primitives with very few subroutine calls. `gl:drawElementsIndirect` behaves similarly to *gl:drawElementsInstancedBaseVertexBaseInstance/7* , execpt that the parameters to *gl:drawElementsInstancedBaseVertexBaseInstance/7* are stored in memory at the address given by `Indirect` .

The parameters addressed by `Indirect` are packed into a structure that takes the form (in C): typedef struct { uint count; uint primCount; uint firstIndex; uint baseVertex; uint baseInstance; } DrawElementsIndirectCommand;

`gl:drawElementsIndirect` is equivalent to: void glDrawElementsIndirect(GLenum mode, GLenum type, const void * indirect) { const DrawElementsIndirectCommand *cmd = (const DrawElementsIndirectCommand *)indirect; glDrawElementsInstancedBaseVertexBaseInstance(mode, cmd->count, type, cmd->firstIndex + size-of-type, cmd->primCount, cmd->baseVertex, cmd->baseInstance); }

If a buffer is bound to the `?GL_DRAW_INDIRECT_BUFFER` binding at the time of a call to `gl:drawElementsIndirect`, `Indirect` is interpreted as an offset, in basic machine units, into that buffer and the parameter data is read from the buffer rather than from client memory.

Note that indices stored in client memory are not supported. If no buffer is bound to the `?GL_ELEMENT_ARRAY_BUFFER` binding, an error will be generated.

The results of the operation are undefined if the reservedMustBeZero member of the parameter structure is non-zero. However, no error is generated in this case.

Vertex attributes that are modified by `gl:drawElementsIndirect` have an unspecified value after `gl:drawElementsIndirect` returns. Attributes that aren't modified remain well defined.

See **external** documentation.

`uniform1d(Location, X) -> ok`

Types:

```
Location = integer()
X = float()
```

See *uniform1f/2*

`uniform2d(Location, X, Y) -> ok`

Types:

```
Location = integer()
X = float()
Y = float()
```

See *uniform1f/2*

uniform3d(Location, X, Y, Z) -> ok

Types:

```
Location = integer()
X = float()
Y = float()
Z = float()
```

See *uniform1f/2*

uniform4d(Location, X, Y, Z, W) -> ok

Types:

```
Location = integer()
X = float()
Y = float()
Z = float()
W = float()
```

See *uniform1f/2*

uniform1dv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [float()]
```

See *uniform1f/2*

uniform2dv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{float(), float()}]
```

See *uniform1f/2*

uniform3dv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{float(), float(), float()}]
```

See *uniform1f/2*

uniform4dv(Location, Value) -> ok

Types:

```
Location = integer()
Value = [{float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix2dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix3dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float()}]
```

See *uniform1f/2*

uniformMatrix4dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *uniform1f/2*

uniformMatrix2x3dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix2x4dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *uniform1f/2*

uniformMatrix3x2dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix3x4dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float()}]
```

See *uniform1f/2*

uniformMatrix4x2dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *uniform1f/2*

uniformMatrix4x3dv(Location, Transpose, Value) -> ok

Types:

```
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float()}]
```

See *uniform1f/2*

getUniformdv(Program, Location) -> matrix()

Types:

```
Program = integer()
Location = integer()
```

See *getUniformfv/2*

getSubroutineUniformLocation(Program, Shadertype, Name) -> integer()

Types:

```
Program = integer()
Shadertype = enum()
Name = string()
```

Retrieve the location of a subroutine uniform of a given shader stage within a program

`gl:getSubroutineUniformLocation` returns the location of the subroutine uniform variable `Name` in the shader stage of type `Shadertype` attached to `Program` , with behavior otherwise identical to *gl:getUniformLocation/2* .

If `Name` is not the name of a subroutine uniform in the shader stage, -1 is returned, but no error is generated. If `Name` is the name of a subroutine uniform in the shader stage, a value between zero and the value of ? `GL_ACTIVE_SUBROUTINE_LOCATIONS` minus one will be returned. Subroutine locations are assigned using

consecutive integers in the range from zero to the value of ?GL_ACTIVE_SUBROUTINE_LOCATIONS minus one for the shader stage. For active subroutine uniforms declared as arrays, the declared array elements are assigned consecutive locations.

See **external** documentation.

getSubroutineIndex(Program, Shadertype, Name) -> integer()

Types:

```
Program = integer()
Shadertype = enum()
Name = string()
```

Retrieve the index of a subroutine uniform of a given shader stage within a program

gl:getSubroutineIndex returns the index of a subroutine uniform within a shader stage attached to a program object. Program contains the name of the program to which the shader is attached. Shadertype specifies the stage from which to query shader subroutine index. Name contains the null-terminated name of the subroutine uniform whose name to query.

If Name is not the name of a subroutine uniform in the shader stage, ?GL_INVALID_INDEX is returned, but no error is generated. If Name is the name of a subroutine uniform in the shader stage, a value between zero and the value of ?GL_ACTIVE_SUBROUTINES minus one will be returned. Subroutine indices are assigned using consecutive integers in the range from zero to the value of ?GL_ACTIVE_SUBROUTINES minus one for the shader stage.

See **external** documentation.

getActiveSubroutineUniformName(Program, Shadertype, Index, Bufsize) -> string()

Types:

```
Program = integer()
Shadertype = enum()
Index = integer()
Bufsize = integer()
```

Query the name of an active shader subroutine uniform

gl:getActiveSubroutineUniformName retrieves the name of an active shader subroutine uniform. Program contains the name of the program containing the uniform. Shadertype specifies the stage for which which the uniform location, given by Index , is valid. Index must be between zero and the value of ?GL_ACTIVE_SUBROUTINE_UNIFORMS minus one for the shader stage.

The uniform name is returned as a null-terminated string in Name . The actual number of characters written into Name , excluding the null terminator is returned in Length . If Length is ?NULL, no length is returned. The maximum number of characters that may be written into Name , including the null terminator, is specified by Bufsize . The length of the longest subroutine uniform name in Program and Shadertype is given by the value of ?GL_ACTIVE_SUBROUTINE_UNIFORM_MAX_LENGTH, which can be queried with *gl:getProgramStageiv/3* .

See **external** documentation.

getActiveSubroutineName(Program, Shadertype, Index, Bufsize) -> string()

Types:

```
Program = integer()
Shadertype = enum()
Index = integer()
```

Bufsize = integer()

Query the name of an active shader subroutine

`gl:getActiveSubroutineName` queries the name of an active shader subroutine uniform from the program object given in `Program` . `Index` specifies the index of the shader subroutine uniform within the shader stage given by `Stage` , and must between zero and the value of `?GL_ACTIVE_SUBROUTINES` minus one for the shader stage.

The name of the selected subroutine is returned as a null-terminated string in `Name` . The actual number of characters written into `Name` , not including the null-terminator, is is returned in `Length` . If `Length` is `?NULL`, no length is returned. The maximum number of characters that may be written into `Name` , including the null-terminator, is given in `Bufsize` .

See **external** documentation.

uniformSubroutinesuiv(Shadertype, Indices) -> ok

Types:

Shadertype = enum()

Indices = [integer()]

Load active subroutine uniforms

`gl:uniformSubroutines` loads all active subroutine uniforms for shader stage `Shadertype` of the current program with subroutine indices from `Indices` , storing `Indices[i]` into the uniform at location `I` . `Count` must be equal to the value of `?GL_ACTIVE_SUBROUTINE_UNIFORM_LOCATIONS` for the program currently in use at shader stage `Shadertype` . Furthermore, all values in `Indices` must be less than the value of `?GL_ACTIVE_SUBROUTINES` for the shader stage.

See **external** documentation.

getUniformSubroutineuiv(Shadertype, Location) -> {integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer(), integer()}

Types:

Shadertype = enum()

Location = integer()

Retrieve the value of a subroutine uniform of a given shader stage of the current program

`gl:getUniformSubroutine` retrieves the value of the subroutine uniform at location `Location` for shader stage `Shadertype` of the current program. `Location` must be less than the value of `?GL_ACTIVE_SUBROUTINE_UNIFORM_LOCATIONS` for the shader currently in use at shader stage `Shadertype` . The value of the subroutine uniform is returned in `Values` .

See **external** documentation.

getProgramStageiv(Program, Shadertype, Pname) -> integer()

Types:

Program = integer()

Shadertype = enum()

Pname = enum()

Retrieve properties of a program object corresponding to a specified shader stage

`gl:getProgramStage` queries a parameter of a shader stage attached to a program object. `Program` contains the name of the program to which the shader is attached. `Shadertype` specifies the stage from which to query the

parameter. Pname specifies which parameter should be queried. The value or values of the parameter to be queried is returned in the variable whose address is given in Values .

If Pname is ?GL_ACTIVE_SUBROUTINE_UNIFORMS, the number of active subroutine variables in the stage is returned in Values .

If Pname is ?GL_ACTIVE_SUBROUTINE_UNIFORM_LOCATIONS, the number of active subroutine variable locations in the stage is returned in Values .

If Pname is ?GL_ACTIVE_SUBROUTINES, the number of active subroutines in the stage is returned in Values .

If Pname is ?GL_ACTIVE_SUBROUTINE_UNIFORM_MAX_LENGTH, the length of the longest subroutine uniform for the stage is returned in Values .

If Pname is ?GL_ACTIVE_SUBROUTINE_MAX_LENGTH, the length of the longest subroutine name for the stage is returned in Values . The returned name length includes space for the null-terminator.

If there is no shader present of type Shadertype , the returned value will be consistent with a shader containing no subroutines or subroutine uniforms.

See **external** documentation.

patchParameteri(Pname, Value) -> ok

Types:

Pname = enum()

Value = integer()

Specifies the parameters for patch primitives

gl:patchParameter specifies the parameters that will be used for patch primitives. Pname specifies the parameter to modify and must be either ?GL_PATCH_VERTICES, ?GL_PATCH_DEFAULT_OUTER_LEVEL or ?GL_PATCH_DEFAULT_INNER_LEVEL. For gl:patchParameteri, Value specifies the new value for the parameter specified by Pname . For gl:patchParameterfv, Values specifies the address of an array containing the new values for the parameter specified by Pname .

When Pname is ?GL_PATCH_VERTICES, Value specifies the number of vertices that will be used to make up a single patch primitive. Patch primitives are consumed by the tessellation control shader (if present) and subsequently used for tessellation. When primitives are specified using *gl:drawArrays/3* or a similar function, each patch will be made from Parameter control points, each represented by a vertex taken from the enabeld vertex arrays. Parameter must be greater than zero, and less than or equal to the value of ?GL_MAX_PATCH_VERTICES.

When Pname is ?GL_PATCH_DEFAULT_OUTER_LEVEL or ?GL_PATCH_DEFAULT_INNER_LEVEL , Values contains the address of an array contiaining the default outer or inner tessellation levels, respectively, to be used when no tessellation control shader is present.

See **external** documentation.

patchParameterfv(Pname, Values) -> ok

Types:

Pname = enum()

Values = [float()]

See *patchParameteri/2*

bindTransformFeedback(Target, Id) -> ok

Types:

Target = enum()

Id = integer()

Bind a transform feedback object

`gl:bindTransformFeedback` binds the transform feedback object with name `Id` to the current GL state. `Id` must be a name previously returned from a call to *gl:genTransformFeedbacks/1* . If `Id` has not previously been bound, a new transform feedback object with name `Id` and initialized with with the default transform state vector is created.

In the initial state, a default transform feedback object is bound and treated as a transform feedback object with a name of zero. If the name zero is subsequently bound, the default transform feedback object is again bound to the GL state.

While a transform feedback buffer object is bound, GL operations on the target to which it is bound affect the bound transform feedback object, and queries of the target to which a transform feedback object is bound return state from the bound object. When buffer objects are bound for transform feedback, they are attached to the currently bound transform feedback object. Buffer objects are used for trans- form feedback only if they are attached to the currently bound transform feedback object.

See **external** documentation.

deleteTransformFeedbacks(Ids) -> ok

Types:

Ids = [integer()]

Delete transform feedback objects

`gl:deleteTransformFeedbacks` deletes the `N` transform feedback objects whose names are stored in the array `Ids` . Unused names in `Ids` are ignored, as is the name zero. After a transform feedback object is deleted, its name is again unused and it has no contents. If an active transform feedback object is deleted, its name immediately becomes unused, but the underlying object is not deleted until it is no longer active.

See **external** documentation.

genTransformFeedbacks(N) -> [integer()]

Types:

N = integer()

Reserve transform feedback object names

`gl:genTransformFeedbacks` returns `N` previously unused transform feedback object names in `Ids` . These names are marked as used, for the purposes of `gl:genTransformFeedbacks` only, but they acquire transform feedback state only when they are first bound.

See **external** documentation.

isTransformFeedback(Id) -> 0 | 1

Types:

Id = integer()

Determine if a name corresponds to a transform feedback object

`gl:isTransformFeedback` returns `?GL_TRUE` if `Id` is currently the name of a transform feedback object. If `Id` is zero, or if `?id` is not the name of a transform feedback object, or if an error occurs, `gl:isTransformFeedback` returns `?GL_FALSE`. If `Id` is a name returned by *gl:genTransformFeedbacks/1* , but that has not yet been bound through a call to *gl:bindTransformFeedback/2* , then the name is not a transform feedback object and `gl:isTransformFeedback` returns `?GL_FALSE` .

See **external** documentation.

pauseTransformFeedback() -> ok

Pause transform feedback operations

`gl:pauseTransformFeedback` pauses transform feedback operations on the currently active transform feedback object. When transform feedback operations are paused, transform feedback is still considered active and changing most transform feedback state related to the object results in an error. However, a new transform feedback object may be bound while transform feedback is paused.

See **external** documentation.

resumeTransformFeedback() -> ok

Resume transform feedback operations

`gl:resumeTransformFeedback` resumes transform feedback operations on the currently active transform feedback object. When transform feedback operations are paused, transform feedback is still considered active and changing most transform feedback state related to the object results in an error. However, a new transform feedback object may be bound while transform feedback is paused.

See **external** documentation.

drawTransformFeedback(Mode, Id) -> ok

Types:

```
Mode = enum()
Id = integer()
```

Render primitives using a count derived from a transform feedback object

`gl:drawTransformFeedback` draws primitives of a type specified by `Mode` using a count retrieved from the transform feedback specified by `Id` . Calling `gl:drawTransformFeedback` is equivalent to calling *gl:drawArrays/3* with `Mode` as specified, `First` set to zero, and `Count` set to the number of vertices captured on vertex stream zero the last time transform feedback was active on the transform feedback object named by `Id` .

See **external** documentation.

drawTransformFeedbackStream(Mode, Id, Stream) -> ok

Types:

```
Mode = enum()
Id = integer()
Stream = integer()
```

Render primitives using a count derived from a specifed stream of a transform feedback object

`gl:drawTransformFeedbackStream` draws primitives of a type specified by `Mode` using a count retrieved from the transform feedback stream specified by `Stream` of the transform feedback object specified by `Id` . Calling `gl:drawTransformFeedbackStream` is equivalent to calling *gl:drawArrays/3* with `Mode` as specified, `First` set to zero, and `Count` set to the number of vertices captured on vertex stream `Stream` the last time transform feedback was active on the transform feedback object named by `Id` .

Calling *gl:drawTransformFeedback/2* is equivalent to calling `gl:drawTransformFeedbackStream` with `Stream` set to zero.

See **external** documentation.

beginQueryIndexed(Target, Index, Id) -> ok

Types:

```
Target = enum()
Index = integer()
Id = integer()
```

glBeginQueryIndexe

See **external** documentation.

endQueryIndexed(Target, Index) -> ok

Types:

```
Target = enum()
Index = integer()
```

Delimit the boundaries of a query object on an indexed target

`gl:beginQueryIndexed` and *gl:endQueryIndexed/2* delimit the boundaries of a query object. `Query` must be a name previously returned from a call to *gl:genQueries/1* . If a query object with name `Id` does not yet exist it is created with the type determined by `Target` . `Target` must be one of `?GL_SAMPLES_PASSED`, `?GL_ANY_SAMPLES_PASSED` , `?GL_PRIMITIVES_GENERATED`, `?GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN`, or `?GL_TIME_ELAPSED` . The behavior of the query object depends on its type and is as follows.

`Index` specifies the index of the query target and must be between a `Target` -specific maximum.

If `Target` is `?GL_SAMPLES_PASSED`, `Id` must be an unused name, or the name of an existing occlusion query object. When `gl:beginQueryIndexed` is executed, the query object's samples-passed counter is reset to 0. Subsequent rendering will increment the counter for every sample that passes the depth test. If the value of `?GL_SAMPLE_BUFFERS` is 0, then the samples-passed count is incremented by 1 for each fragment. If the value of `?GL_SAMPLE_BUFFERS` is 1, then the samples-passed count is incremented by the number of samples whose coverage bit is set. However, implementations, at their discression may instead increase the samples-passed count by the value of `?GL_SAMPLES` if any sample in the fragment is covered. When `gl:endQueryIndexed` is executed, the samples-passed counter is assigned to the query object's result value. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`. When `Target` is `?GL_SAMPLES_PASSED`, `Index` must be zero.

If `Target` is `?GL_ANY_SAMPLES_PASSED`, `Id` must be an unused name, or the name of an existing boolean occlusion query object. When `gl:beginQueryIndexed` is executed, the query object's samples-passed flag is reset to `?GL_FALSE`. Subsequent rendering causes the flag to be set to `?GL_TRUE` if any sample passes the depth test. When `gl:endQueryIndexed` is executed, the samples-passed flag is assigned to the query object's result value. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`. When `Target` is `?GL_ANY_SAMPLES_PASSED` , `Index` must be zero.

If `Target` is `?GL_PRIMITIVES_GENERATED`, `Id` must be an unused name, or the name of an existing primitive query object previously bound to the `?GL_PRIMITIVES_GENERATED` query binding. When `gl:beginQueryIndexed` is executed, the query object's primitives-generated counter is reset to 0. Subsequent rendering will increment the counter once for every vertex that is emitted from the geometry shader to the stream given by `Index` , or from the vertex shader if `Index` is zero and no geometry shader is present. When `gl:endQueryIndexed` is executed, the primitives-generated counter for stream `Index` is assigned to the query object's result value. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`. When `Target` is `?GL_PRIMITIVES_GENERATED` , `Index` must be less than the value of `?GL_MAX_VERTEX_STREAMS`.

If `Target` is `?GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN`, `Id` must be an unused name, or the name of an existing primitive query object previously bound to the `?GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN` query binding. When `gl:beginQueryIndexed` is executed, the query object's primitives-written counter for the stream specified by `Index` is reset to 0. Subsequent rendering will increment the counter once for every vertex that is written into the bound

transform feedback buffer(s) for stream `Index` . If transform feedback mode is not activated between the call to `gl:beginQueryIndexed` and `gl:endQueryIndexed`, the counter will not be incremented. When `gl:endQueryIndexed` is executed, the primitives-written counter for stream `Index` is assigned to the query object's result value. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`. When `Target` is `?GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN` , `Index` must be less than the value of `?GL_MAX_VERTEX_STREAMS`.

If `Target` is `?GL_TIME_ELAPSED`, `Id` must be an unused name, or the name of an existing timer query object previously bound to the `?GL_TIME_ELAPSED` query binding. When `gl:beginQueryIndexed` is executed, the query object's time counter is reset to 0. When `gl:endQueryIndexed` is executed, the elapsed server time that has passed since the call to `gl:beginQueryIndexed` is written into the query object's time counter. This value can be queried by calling *gl:getQueryObjectiv/2* with `Pname` `?GL_QUERY_RESULT`. When `Target` is `?GL_TIME_ELAPSED`, `Index` must be zero.

Querying the `?GL_QUERY_RESULT` implicitly flushes the GL pipeline until the rendering delimited by the query object has completed and the result is available. `?GL_QUERY_RESULT_AVAILABLE` can be queried to determine if the result is immediately available or if the rendering is not yet complete.

See **external** documentation.

`getQueryIndexediv(Target, Index, Pname) -> integer()`

Types:

```
Target = enum()
Index = integer()
Pname = enum()
```

Return parameters of an indexed query object target

`gl:getQueryIndexediv` returns in `Params` a selected parameter of the indexed query object target specified by `Target` and `Index` . `Index` specifies the index of the query object target and must be between zero and a target-specific maxiumum.

`Pname` names a specific query object target parameter. When `Pname` is `?GL_CURRENT_QUERY` , the name of the currently active query for the specified `Index` of `Target` , or zero if no query is active, will be placed in `Params` . If `Pname` is `?GL_QUERY_COUNTER_BITS` , the implementation-dependent number of bits used to hold the result of queries for `Target` is returned in `Params` .

See **external** documentation.

`releaseShaderCompiler() -> ok`

Release resources consumed by the implementation's shader compiler

`gl:releaseShaderCompiler` provides a hint to the implementation that it may free internal resources associated with its shader compiler. *gl:compileShader/1* may subsequently be called and the implementation may at that time reallocate resources previously freed by the call to `gl:releaseShaderCompiler`.

See **external** documentation.

`shaderBinary(Shaders, Binaryformat, Binary) -> ok`

Types:

```
Shaders = [integer()]
Binaryformat = enum()
Binary = binary()
```

Load pre-compiled shader binaries

gl:shaderBinary loads pre-compiled shader binary code into the Count shader objects whose handles are given in Shaders . Binary points to Length bytes of binary shader code stored in client memory. BinaryFormat specifies the format of the pre-compiled code.

The binary image contained in Binary will be decoded according to the extension specification defining the specified BinaryFormat token. OpenGL does not define any specific binary formats, but it does provide a mechanism to obtain token vaues for such formats provided by such extensions.

Depending on the types of the shader objects in Shaders , gl:shaderBinary will individually load binary vertex or fragment shaders, or load an executable binary that contains an optimized pair of vertex and fragment shaders stored in the same binary.

See **external** documentation.

getShaderPrecisionFormat(Shadertype, Precisiontype) -> {Range::{integer(), integer()}, Precision::integer()}

Types:

Shadertype = enum()

Precisiontype = enum()

Retrieve the range and precision for numeric formats supported by the shader compiler

gl:getShaderPrecisionFormat retrieves the numeric range and precision for the implementation's representation of quantities in different numeric formats in specified shader type. ShaderType specifies the type of shader for which the numeric precision and range is to be retrieved and must be one of ?GL_VERTEX_SHADER or ?GL_FRAGMENT_SHADER. PrecisionType specifies the numeric format to query and must be one of ?GL_LOW_FLOAT, ?GL_MEDIUM_FLOAT ?GL_HIGH_FLOAT, ?GL_LOW_INT, ?GL_MEDIUM_INT, or ?GL_HIGH_INT.

Range points to an array of two integers into which the format's numeric range will be returned. If min and max are the smallest values representable in the format, then the values returned are defined to be: Range [0] = floor(log2(|min|)) and Range [1] = floor(log2(|max|)).

Precision specifies the address of an integer into which will be written the log2 value of the number of bits of precision of the format. If the smallest representable value greater than 1 is 1 + eps, then the integer addressed by Precision will contain floor(-log2(eps)).

See **external** documentation.

depthRangef(N, F) -> ok

Types:

N = clamp()

F = clamp()

See *depthRange/2*

clearDepthf(D) -> ok

Types:

D = clamp()

glClearDepthf

See **external** documentation.

getProgramBinary(Program, BufSize) -> {BinaryFormat::enum(), Binary::binary()}

Types:

Program = integer()

BufSize = integer()

Return a binary representation of a program object's compiled and linked executable source

`gl:getProgramBinary` returns a binary representation of the compiled and linked executable for `Program` into the array of bytes whose address is specified in `Binary` . The maximum number of bytes that may be written into `Binary` is specified by `BufSize` . If the program binary is greater in size than `BufSize` bytes, then an error is generated, otherwise the actual number of bytes written into `Binary` is returned in the variable whose address is given by `Length` . If `Length` is `?NULL`, then no length is returned.

The format of the program binary written into `Binary` is returned in the variable whose address is given by `BinaryFormat` , and may be implementation dependent. The binary produced by the GL may subsequently be returned to the GL by calling *gl:programBinary/3* , with `BinaryFormat` and `Length` set to the values returned by `gl:getProgramBinary` , and passing the returned binary data in the `Binary` parameter.

See **external** documentation.

programBinary(Program, BinaryFormat, Binary) -> ok

Types:

Program = integer()

BinaryFormat = enum()

Binary = binary()

Load a program object with a program binary

`gl:programBinary` loads a program object with a program binary previously returned from *gl:getProgramBinary/2* . `BinaryFormat` and `Binary` must be those returned by a previous call to *gl:getProgramBinary/2* , and `Length` must be the length returned by *gl:getProgramBinary/2* , or by *gl:getProgramiv/2* when called with `Pname` set to `?GL_PROGRAM_BINARY_LENGTH`. If these conditions are not met, loading the program binary will fail and `Program` 's `?GL_LINK_STATUS` will be set to `?GL_FALSE`.

A program object's program binary is replaced by calls to *gl:linkProgram/1* or `gl:programBinary` . When linking success or failure is concerned, `gl:programBinary` can be considered to perform an implicit linking operation. *gl:linkProgram/1* and `gl:programBinary` both set the program object's `?GL_LINK_STATUS` to `?GL_TRUE` or `?GL_FALSE` .

A successful call to `gl:programBinary` will reset all uniform variables to their initial values. The initial value is either the value of the variable's initializer as specified in the original shader source, or zero if no initializer was present. Additionally, all vertex shader input and fragment shader output assignments that were in effect when the program was linked before saving are restored with `gl:programBinary` is called.

See **external** documentation.

programParameteri(Program, Pname, Value) -> ok

Types:

Program = integer()

Pname = enum()

Value = integer()

Specify a parameter for a program object

gl:programParameter specifies a new value for the parameter nameed by Pname for the program object Program .

If Pname is ?GL_PROGRAM_BINARY_RETRIEVABLE_HINT, Value should be ?GL_FALSE or ?GL_TRUE to indicate to the implementation the intention of the application to retrieve the program's binary representation with *gl:getProgramBinary/2* . The implementation may use this information to store information that may be useful for a future query of the program's binary. It is recommended to set ?GL_PROGRAM_BINARY_RETRIEVABLE_HINT for the program to ?GL_TRUE before calling *gl:linkProgram/1* , and using the program at run-time if the binary is to be retrieved later.

If Pname is ?GL_PROGRAM_SEPARABLE, Value must be ?GL_TRUE or ?GL_FALSE and indicates whether Program can be bound to individual pipeline stages via *gl:useProgramStages/3* . A program's ?GL_PROGRAM_SEPARABLE parameter must be set to ?GL_TRUEbefore *gl:linkProgram/1* is called in order for it to be usable with a program pipeline object. The initial state of ?GL_PROGRAM_SEPARABLE is ?GL_FALSE.

See **external** documentation.

useProgramStages(Pipeline, Stages, Program) -> ok

Types:

Pipeline = integer()
Stages = integer()
Program = integer()

Bind stages of a program object to a program pipeline

gl:useProgramStages binds executables from a program object associated with a specified set of shader stages to the program pipeline object given by Pipeline . Pipeline specifies the program pipeline object to which to bind the executables. Stages contains a logical combination of bits indicating the shader stages to use within Program with the program pipeline object Pipeline . Stages must be a logical combination of ?GL_VERTEX_SHADER_BIT, ?GL_TESS_CONTROL_SHADER_BIT, ?GL_TESS_EVALUATION_SHADER_BIT , ?GL_GEOMETRY_SHADER_BIT, and ?GL_FRAGMENT_SHADER_BIT. Additionally, the special value ?GL_ALL_SHADER_BITS may be specified to indicate that all executables contained in Program should be installed in Pipeline .

If Program refers to a program object with a valid shader attached for an indicated shader stage, gl:useProgramStages installs the executable code for that stage in the indicated program pipeline object Pipeline . If Program is zero, or refers to a program object with no valid shader executable for a given stage, it is as if the pipeline object has no programmable stage configured for the indicated shader stages. If Stages contains bits other than those listed above, and is not equal to ?GL_ALL_SHADER_BITS , an error is generated.

See **external** documentation.

activeShaderProgram(Pipeline, Program) -> ok

Types:

Pipeline = integer()
Program = integer()

Set the active program object for a program pipeline object

gl:activeShaderProgram sets the linked program named by Program to be the active program for the program pipeline object Pipeline . The active program in the active program pipeline object is the target of calls to *gl:uniform1f/2* when no program has been made current through a call to *gl:useProgram/1* .

See **external** documentation.

createShaderProgramv(Type, Strings) -> integer()

Types:

Type = enum()

Strings = [string()]

glCreateShaderProgramv

See **external** documentation.

bindProgramPipeline(Pipeline) -> ok

Types:

Pipeline = integer()

Bind a program pipeline to the current context

`gl:bindProgramPipeline` binds a program pipeline object to the current context. `Pipeline` must be a name previously returned from a call to *gl:genProgramPipelines/1* . If no program pipeline exists with name `Pipeline` then a new pipeline object is created with that name and initialized to the default state vector.

When a program pipeline object is bound using `gl:bindProgramPipeline`, any previous binding is broken and is replaced with a binding to the specified pipeline object. If `Pipeline` is zero, the previous binding is broken and is not replaced, leaving no pipeline object bound. If no current program object has been established by *gl:useProgram/1* , the program objects used for each stage and for uniform updates are taken from the bound program pipeline object, if any. If there is a current program object established by *gl:useProgram/1* , the bound program pipeline object has no effect on rendering or uniform updates. When a bound program pipeline object is used for rendering, individual shader executables are taken from its program objects.

See **external** documentation.

deleteProgramPipelines(Pipelines) -> ok

Types:

Pipelines = [integer()]

Delete program pipeline objects

`gl:deleteProgramPipelines` deletes the `N` program pipeline objects whose names are stored in the array `Pipelines` . Unused names in `Pipelines` are ignored, as is the name zero. After a program pipeline object is deleted, its name is again unused and it has no contents. If program pipeline object that is currently bound is deleted, the binding for that object reverts to zero and no program pipeline object becomes current.

See **external** documentation.

genProgramPipelines(N) -> [integer()]

Types:

N = integer()

Reserve program pipeline object names

`gl:genProgramPipelines` returns `N` previously unused program pipeline object names in `Pipelines` . These names are marked as used, for the purposes of `gl:genProgramPipelines` only, but they acquire program pipeline state only when they are first bound.

See **external** documentation.

isProgramPipeline(Pipeline) -> 0 | 1

Types:

Pipeline = integer()

Determine if a name corresponds to a program pipeline object

`gl:isProgramPipeline` returns `?GL_TRUE` if `Pipeline` is currently the name of a program pipeline object. If `Pipeline` is zero, or if `?pipeline` is not the name of a program pipeline object, or if an error occurs, `gl:isProgramPipeline` returns `?GL_FALSE`. If `Pipeline` is a name returned by *gl:genProgramPipelines/1* , but that has not yet been bound through a call to *gl:bindProgramPipeline/1* , then the name is not a program pipeline object and `gl:isProgramPipeline` returns `?GL_FALSE` .

See **external** documentation.

getProgramPipelineiv(Pipeline, Pname) -> integer()

Types:

Pipeline = integer()

Pname = enum()

Retrieve properties of a program pipeline object

`gl:getProgramPipelineiv` retrieves the value of a property of the program pipeline object `Pipeline` . `Pname` specifies the name of the parameter whose value to retrieve. The value of the parameter is written to the variable whose address is given by `Params` .

If `Pname` is `?GL_ACTIVE_PROGRAM`, the name of the active program object of the program pipeline object is returned in `Params` .

If `Pname` is `?GL_VERTEX_SHADER`, the name of the current program object for the vertex shader type of the program pipeline object is returned in `Params` .

If `Pname` is `?GL_TESS_CONTROL_SHADER`, the name of the current program object for the tessellation control shader type of the program pipeline object is returned in `Params` .

If `Pname` is `?GL_TESS_EVALUATION_SHADER`, the name of the current program object for the tessellation evaluation shader type of the program pipeline object is returned in `Params` .

If `Pname` is `?GL_GEOMETRY_SHADER`, the name of the current program object for the geometry shader type of the program pipeline object is returned in `Params` .

If `Pname` is `?GL_FRAGMENT_SHADER`, the name of the current program object for the fragment shader type of the program pipeline object is returned in `Params` .

If `Pname` is `?GL_INFO_LOG_LENGTH`, the length of the info log, including the null terminator, is returned in `Params` . If there is no info log, zero is returned.

See **external** documentation.

programUniform1i(Program, Location, V0) -> ok

Types:

Program = integer()

Location = integer()

V0 = integer()

Specify the value of a uniform variable for a specified program object

`gl:programUniform` modifies the value of a uniform variable or a uniform variable array. The location of the uniform variable to be modified is specified by `Location` , which should be a value returned by *gl:getUniformLocation/2* . `gl:programUniform` operates on the program object specified by `Program` .

The commands `gl:programUniform{1|2|3|4}{f|i|ui}` are used to change the value of the uniform variable specified by `Location` using the values passed as arguments. The number specified in the command should match the number of components in the data type of the specified uniform variable (e.g., `1` for float, int, unsigned int, bool; `2` for vec2, ivec2, uvec2, bvec2, etc.). The suffix `f` indicates that floating-point values are being passed; the suffix `i` indicates that integer values are being passed; the suffix `ui` indicates that unsigned integer values are being passed, and this type should also match the data type of the specified uniform variable. The `i` variants of this function should be used to provide values for uniform variables defined as int, ivec2 , ivec3, ivec4, or arrays of these. The `ui` variants of this function should be used to provide values for uniform variables defined as unsigned int, uvec2, uvec3, uvec4, or arrays of these. The `f` variants should be used to provide values for uniform variables of type float, vec2, vec3, vec4, or arrays of these. Either the `i`, `ui` or `f` variants may be used to provide values for uniform variables of type bool, bvec2 , bvec3, bvec4, or arrays of these. The uniform variable will be set to false if the input value is 0 or 0.0f, and it will be set to true otherwise.

All active uniform variables defined in a program object are initialized to 0 when the program object is linked successfully. They retain the values assigned to them by a call to `gl:programUniform` until the next successful link operation occurs on the program object, when they are once again initialized to 0.

The commands `gl:programUniform{1|2|3|4}{f|i|ui}v` can be used to modify a single uniform variable or a uniform variable array. These commands pass a count and a pointer to the values to be loaded into a uniform variable or a uniform variable array. A count of 1 should be used if modifying the value of a single uniform variable, and a count of 1 or greater can be used to modify an entire array or part of an array. When loading `n` elements starting at an arbitrary position `m` in a uniform variable array, elements `m` + `n` - 1 in the array will be replaced with the new values. If `M` + `N` - 1 is larger than the size of the uniform variable array, values for all array elements beyond the end of the array will be ignored. The number specified in the name of the command indicates the number of components for each element in `Value` , and it should match the number of components in the data type of the specified uniform variable (e.g., `1` for float, int, bool; `2` for vec2, ivec2, bvec2, etc.). The data type specified in the name of the command must match the data type for the specified uniform variable as described previously for `gl:programUniform{1|2|3|4}{f|i|ui}`.

For uniform variable arrays, each element of the array is considered to be of the type indicated in the name of the command (e.g., `gl:programUniform3f` or `gl:programUniform3fv` can be used to load a uniform variable array of type vec3). The number of elements of the uniform variable array to be modified is specified by `Count`

The commands `gl:programUniformMatrix{2|3|4|2x3|3x2|2x4|4x2|3x4|4x3}fv` are used to modify a matrix or an array of matrices. The numbers in the command name are interpreted as the dimensionality of the matrix. The number `2` indicates a 2 × 2 matrix (i.e., 4 values), the number `3` indicates a 3 × 3 matrix (i.e., 9 values), and the number `4` indicates a 4 × 4 matrix (i.e., 16 values). Non-square matrix dimensionality is explicit, with the first number representing the number of columns and the second number representing the number of rows. For example, `2x4` indicates a 2 × 4 matrix with 2 columns and 4 rows (i.e., 8 values). If `Transpose` is `?GL_FALSE`, each matrix is assumed to be supplied in column major order. If `Transpose` is `?GL_TRUE`, each matrix is assumed to be supplied in row major order. The `Count` argument indicates the number of matrices to be passed. A count of 1 should be used if modifying the value of a single matrix, and a count greater than 1 can be used to modify an array of matrices.

See **external** documentation.

`programUniform1iv(Program, Location, Value) -> ok`

Types:

```
Program = integer()
Location = integer()
Value = [integer()]
```

See *programUniform1i/3*

programUniform1f(Program, Location, V0) -> ok

Types:

Program = integer()
Location = integer()
V0 = float()

See *programUniform1i/3*

programUniform1fv(Program, Location, Value) -> ok

Types:

Program = integer()
Location = integer()
Value = [float()]

See *programUniform1i/3*

programUniform1d(Program, Location, V0) -> ok

Types:

Program = integer()
Location = integer()
V0 = float()

See *programUniform1i/3*

programUniform1dv(Program, Location, Value) -> ok

Types:

Program = integer()
Location = integer()
Value = [float()]

See *programUniform1i/3*

programUniform1ui(Program, Location, V0) -> ok

Types:

Program = integer()
Location = integer()
V0 = integer()

See *programUniform1i/3*

programUniform1uiv(Program, Location, Value) -> ok

Types:

Program = integer()
Location = integer()
Value = [integer()]

See *programUniform1i/3*

programUniform2i(Program, Location, V0, V1) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = integer()
V1 = integer()
```

See *programUniform1i/3*

programUniform2iv(Program, Location, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Value = [{integer(), integer()}]
```

See *programUniform1i/3*

programUniform2f(Program, Location, V0, V1) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = float()
V1 = float()
```

See *programUniform1i/3*

programUniform2fv(Program, Location, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Value = [{float(), float()}]
```

See *programUniform1i/3*

programUniform2d(Program, Location, V0, V1) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = float()
V1 = float()
```

See *programUniform1i/3*

programUniform2dv(Program, Location, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Value = [{float(), float()}]
```

See *programUniform1i/3*

programUniform2ui(Program, Location, V0, V1) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = integer()
V1 = integer()
```

See *programUniform1i/3*

programUniform2uiv(Program, Location, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Value = [{integer(), integer()}]
```

See *programUniform1i/3*

programUniform3i(Program, Location, V0, V1, V2) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = integer()
V1 = integer()
V2 = integer()
```

See *programUniform1i/3*

programUniform3iv(Program, Location, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Value = [{integer(), integer(), integer()}]
```

See *programUniform1i/3*

programUniform3f(Program, Location, V0, V1, V2) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = float()
V1 = float()
V2 = float()
```

See *programUniform1i/3*

programUniform3fv(Program, Location, Value) -> ok

Types:

Program = integer()
Location = integer()
Value = [{float(), float(), float()}]

See *programUniform1i/3*

programUniform3d(Program, Location, V0, V1, V2) -> ok

Types:

Program = integer()
Location = integer()
V0 = float()
V1 = float()
V2 = float()

See *programUniform1i/3*

programUniform3dv(Program, Location, Value) -> ok

Types:

Program = integer()
Location = integer()
Value = [{float(), float(), float()}]

See *programUniform1i/3*

programUniform3ui(Program, Location, V0, V1, V2) -> ok

Types:

Program = integer()
Location = integer()
V0 = integer()
V1 = integer()
V2 = integer()

See *programUniform1i/3*

programUniform3uiv(Program, Location, Value) -> ok

Types:

Program = integer()
Location = integer()
Value = [{integer(), integer(), integer()}]

See *programUniform1i/3*

programUniform4i(Program, Location, V0, V1, V2, V3) -> ok

Types:

Program = integer()
Location = integer()

```
V0 = integer()
V1 = integer()
V2 = integer()
V3 = integer()
```

See *programUniform1i/3*

programUniform4iv(Program, Location, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Value = [{integer(), integer(), integer(), integer()}]
```

See *programUniform1i/3*

programUniform4f(Program, Location, V0, V1, V2, V3) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = float()
V1 = float()
V2 = float()
V3 = float()
```

See *programUniform1i/3*

programUniform4fv(Program, Location, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Value = [{float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniform4d(Program, Location, V0, V1, V2, V3) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = float()
V1 = float()
V2 = float()
V3 = float()
```

See *programUniform1i/3*

programUniform4dv(Program, Location, Value) -> ok

Types:

```
Program = integer()
```

```
Location = integer()
Value = [{float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniform4ui(Program, Location, V0, V1, V2, V3) -> ok

Types:

```
Program = integer()
Location = integer()
V0 = integer()
V1 = integer()
V2 = integer()
V3 = integer()
```

See *programUniform1i/3*

programUniform4uiv(Program, Location, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Value = [{integer(), integer(), integer(), integer()}]
```

See *programUniform1i/3*

programUniformMatrix2fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix3fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(),
float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix4fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
```

```
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *programUniform1i/3*

programUniformMatrix2dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix3dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix4dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *programUniform1i/3*

programUniformMatrix2x3fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix3x2fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
```

```
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix2x4fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *programUniform1i/3*

programUniformMatrix4x2fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *programUniform1i/3*

programUniformMatrix3x4fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix4x3fv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix2x3dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
```

```
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix3x2dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix2x4dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *programUniform1i/3*

programUniformMatrix4x2dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float()}]
```

See *programUniform1i/3*

programUniformMatrix3x4dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float()}]
```

See *programUniform1i/3*

programUniformMatrix4x3dv(Program, Location, Transpose, Value) -> ok

Types:

```
Program = integer()
Location = integer()
```

```
Transpose = 0 | 1
Value = [{float(), float(), float(), float(), float(), float(), float(),
float(), float(), float(), float(), float()}]
```

See *programUniform1i/3*

validateProgramPipeline(Pipeline) -> ok

Types:

```
Pipeline = integer()
```

Validate a program pipeline object against current GL state

`gl:validateProgramPipeline` instructs the implementation to validate the shader executables contained in `Pipeline` against the current GL state. The implementation may use this as an opportunity to perform any internal shader modifications that may be required to ensure correct operation of the installed shaders given the current GL state.

After a program pipeline has been validated, its validation status is set to `?GL_TRUE` . The validation status of a program pipeline object may be queried by calling *gl:getProgramPipelineiv/2* with parameter `?GL_VALIDATE_STATUS`.

If `Pipeline` is a name previously returned from a call to *gl:genProgramPipelines/1* but that has not yet been bound by a call to *gl:bindProgramPipeline/1* , a new program pipeline object is created with name `Pipeline` and the default state vector.

See **external** documentation.

getProgramPipelineInfoLog(Pipeline, BufSize) -> string()

Types:

```
Pipeline = integer()
BufSize = integer()
```

Retrieve the info log string from a program pipeline object

`gl:getProgramPipelineInfoLog` retrieves the info log for the program pipeline object `Pipeline` . The info log, including its null terminator, is written into the array of characters whose address is given by `InfoLog` . The maximum number of characters that may be written into `InfoLog` is given by `BufSize` , and the actual number of characters written into `InfoLog` is returned in the integer whose address is given by `Length` . If `Length` is `?NULL`, no length is returned.

The actual length of the info log for the program pipeline may be determined by calling *gl:getProgramPipelineiv/2* with `Pname` set to `?GL_INFO_LOG_LENGTH`.

See **external** documentation.

vertexAttribL1d(Index, X) -> ok

Types:

```
Index = integer()
X = float()
```

glVertexAttribL

See **external** documentation.

vertexAttribL2d(Index, X, Y) -> ok

Types:

```
Index = integer()
```

```
X = float()
Y = float()
```

glVertexAttribL

See **external** documentation.

vertexAttribL3d(Index, X, Y, Z) -> ok

Types:

```
Index = integer()
X = float()
Y = float()
Z = float()
```

glVertexAttribL

See **external** documentation.

vertexAttribL4d(Index, X, Y, Z, W) -> ok

Types:

```
Index = integer()
X = float()
Y = float()
Z = float()
W = float()
```

glVertexAttribL

See **external** documentation.

vertexAttribL1dv(Index::integer(), V) -> ok

Types:

```
V = {X::float()}
```

Equivalent to *vertexAttribL1d(Index, X)*.

vertexAttribL2dv(Index::integer(), V) -> ok

Types:

```
V = {X::float(), Y::float()}
```

Equivalent to *vertexAttribL2d(Index, X, Y)*.

vertexAttribL3dv(Index::integer(), V) -> ok

Types:

```
V = {X::float(), Y::float(), Z::float()}
```

Equivalent to *vertexAttribL3d(Index, X, Y, Z)*.

vertexAttribL4dv(Index::integer(), V) -> ok

Types:

```
V = {X::float(), Y::float(), Z::float(), W::float()}
```

Equivalent to *vertexAttribL4d(Index, X, Y, Z, W)*.

vertexAttribLPointer(Index, Size, Type, Stride, Pointer) -> ok

Types:

```
Index = integer()
Size = integer()
Type = enum()
Stride = integer()
Pointer = offset() | mem()
```

glVertexAttribLPointer

See **external** documentation.

getVertexAttribLdv(Index, Pname) -> {float(), float(), float(), float()}

Types:

```
Index = integer()
Pname = enum()
```

glGetVertexAttribL

See **external** documentation.

viewportArrayv(First, V) -> ok

Types:

```
First = integer()
V = [{float(), float(), float(), float()}]
```

glViewportArrayv

See **external** documentation.

viewportIndexedf(Index, X, Y, W, H) -> ok

Types:

```
Index = integer()
X = float()
Y = float()
W = float()
H = float()
```

Set a specified viewport

`gl:viewportIndexedf` and `gl:viewportIndexedfv` specify the parameters for a single viewport. `Index` specifies the index of the viewport to modify. `Index` must be less than the value of `?GL_MAX_VIEWPORTS`. For `gl:viewportIndexedf`, `X` , `Y` , `W` , and `H` specify the left, bottom, width and height of the viewport in pixels, respectively. For `gl:viewportIndexedfv`, `V` contains the address of an array of floating point values specifying the left (x), bottom (y), width (w), and height (h) of each viewport, in that order. x and y give the location of the viewport's lower left corner, and w and h give the width and height of the viewport, respectively. The viewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let (x nd y nd) be normalized device coordinates. Then the window coordinates (x w y w) are computed as follows:

x w=(x nd+1) (width/2)+x

y w=(y nd+1) (height/2)+y

The location of the viewport's bottom left corner, given by (x, y) is clamped to be within the implementaiton-dependent viewport bounds range. The viewport bounds range [min, max] can be determined by calling *gl:getBooleanv/1* with argument `?GL_VIEWPORT_BOUNDS_RANGE` . Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, call *gl:getBooleanv/1* with argument `?GL_MAX_VIEWPORT_DIMS`.

The precision with which the GL interprets the floating point viewport bounds is implementation-dependent and may be determined by querying the impementation-defined constant `?GL_VIEWPORT_SUBPIXEL_BITS` .

Calling `gl:viewportIndexedfv` is equivalent to calling see `glViewportArray` with `First` set to `Index` , `Count` set to 1 and `V` passsed directly. `gl:viewportIndexedf` is equivalent to: void glViewportIndexedf(GLuint index, GLfloat x, GLfloat y, GLfloat w, GLfloat h) { const float v[4] = { x, y, w, h }; glViewportArrayv(index, 1, v); }

See **external** documentation.

`viewportIndexedfv(Index, V) -> ok`

Types:

```
Index = integer()
V = {float(), float(), float(), float()}
```

See *viewportIndexedf/5*

`scissorArrayv(First, V) -> ok`

Types:

```
First = integer()
V = [{integer(), integer(), integer(), integer()}]
```

glScissorArrayv

See **external** documentation.

`scissorIndexed(Index, Left, Bottom, Width, Height) -> ok`

Types:

```
Index = integer()
Left = integer()
Bottom = integer()
Width = integer()
Height = integer()
```

glScissorIndexe

See **external** documentation.

`scissorIndexedv(Index, V) -> ok`

Types:

```
Index = integer()
V = {integer(), integer(), integer(), integer()}
```

glScissorIndexe

See **external** documentation.

depthRangeArrayv(First, V) -> ok

Types:

```
First = integer()
V = [{clamp(), clamp()}]
```

glDepthRangeArrayv

See **external** documentation.

depthRangeIndexed(Index, N, F) -> ok

Types:

```
Index = integer()
N = clamp()
F = clamp()
```

glDepthRangeIndexe

See **external** documentation.

getFloati_v(Target, Index) -> [float()]

Types:

```
Target = enum()
Index = integer()
```

See *getBooleanv/1*

getDoublei_v(Target, Index) -> [float()]

Types:

```
Target = enum()
Index = integer()
```

See *getBooleanv/1*

debugMessageControlARB(Source, Type, Severity, Ids, Enabled) -> ok

Types:

```
Source = enum()
Type = enum()
Severity = enum()
Ids = [integer()]
Enabled = 0 | 1
```

glDebugMessageControlARB

See **external** documentation.

debugMessageInsertARB(Source, Type, Id, Severity, Buf) -> ok

Types:

```
Source = enum()
Type = enum()
Id = integer()
```

Severity = enum()

Buf = string()

glDebugMessageInsertARB

See **external** documentation.

getDebugMessageLogARB(Count, Bufsize) -> {integer(), Sources::[enum()], Types::[enum()], Ids::[integer()], Severities::[enum()], MessageLog::[string()]}

Types:

Count = integer()

Bufsize = integer()

glGetDebugMessageLogARB

See **external** documentation.

getGraphicsResetStatusARB() -> enum()

glGetGraphicsResetStatusARB

See **external** documentation.

drawArraysInstancedBaseInstance(Mode, First, Count, Primcount, Baseinstance) -> ok

Types:

Mode = enum()

First = integer()

Count = integer()

Primcount = integer()

Baseinstance = integer()

Draw multiple instances of a range of elements with offset applied to instanced attributes

`gl:drawArraysInstancedBaseInstance` behaves identically to *gl:drawArrays/3* except that `Primcount` instances of the range of elements are executed and the value of the internal counter `InstanceID` advances for each iteration. `InstanceID` is an internal 32-bit integer counter that may be read by a vertex shader as `?gl_InstanceID`.

`gl:drawArraysInstancedBaseInstance` has the same effect as: if (mode or count is invalid) generate appropriate error else { for (int i = 0; i < primcount ; i++) { instanceID = i; glDrawArrays(mode, first, count); } instanceID = 0; }

Specific vertex attributes may be classified as `instanced` through the use of *gl:vertexAttribDivisor/2* . Instanced vertex attributes supply per-instance vertex data to the vertex shader. The index of the vertex fetched from the enabled instanced vertex attribute arrays is calculated as: |gl_ InstanceID/divisor|+ baseInstance. Note that `Baseinstance` does not affect the shader-visible value of `?gl_InstanceID`.

See **external** documentation.

drawElementsInstancedBaseInstance(Mode, Count, Type, Indices, Primcount, Baseinstance) -> ok

Types:

Mode = enum()

Count = integer()
Type = enum()
Indices = offset() | mem()
Primcount = integer()
Baseinstance = integer()

Draw multiple instances of a set of elements with offset applied to instanced attributes

`gl:drawElementsInstancedBaseInstance` behaves identically to *gl:drawElements/4* except that `Primcount` instances of the set of elements are executed and the value of the internal counter `InstanceID` advances for each iteration. `InstanceID` is an internal 32-bit integer counter that may be read by a vertex shader as `?gl_InstanceID` .

`gl:drawElementsInstancedBaseInstance` has the same effect as: if (mode, count, or type is invalid) generate appropriate error else { for (int i = 0; i < primcount ; i++) { instanceID = i; glDrawElements(mode, count, type, indices); } instanceID = 0; }

Specific vertex attributes may be classified as `instanced` through the use of *gl:vertexAttribDivisor/2* . Instanced vertex attributes supply per-instance vertex data to the vertex shader. The index of the vertex fetched from the enabled instanced vertex attribute arrays is calculated as |gl_ InstanceID/divisor|+ baseInstance. Note that `Baseinstance` does not affect the shader-visible value of `?gl_InstanceID`.

See **external** documentation.

drawElementsInstancedBaseVertexBaseInstance(Mode, Count, Type, Indices, Primcount, Basevertex, Baseinstance) -> ok

Types:

Mode = enum()
Count = integer()
Type = enum()
Indices = offset() | mem()
Primcount = integer()
Basevertex = integer()
Baseinstance = integer()

Render multiple instances of a set of primitives from array data with a per-element offset

`gl:drawElementsInstancedBaseVertexBaseInstance` behaves identically to *gl:drawElementsInstanced/5* except that the `i`th element transferred by the corresponding draw call will be taken from element `Indices` [i] + `Basevertex` of each enabled array. If the resulting value is larger than the maximum value representable by `Type` , it is as if the calculation were upconverted to 32-bit unsigned integers (with wrapping on overflow conditions). The operation is undefined if the sum would be negative. The `Basevertex` has no effect on the shader-visible value of `?gl_VertexID`.

Specific vertex attributes may be classified as `instanced` through the use of *gl:vertexAttribDivisor/2* . Instanced vertex attributes supply per-instance vertex data to the vertex shader. The index of the vertex fetched from the enabled instanced vertex attribute arrays is calculated as |gl_ InstanceID/divisor|+ baseInstance. Note that `Baseinstance` does not affect the shader-visible value of `?gl_InstanceID`.

See **external** documentation.

drawTransformFeedbackInstanced(Mode, Id, Primcount) -> ok

Types:

Mode = enum()

```
Id = integer()
Primcount = integer()
```

glDrawTransformFeedbackInstance

See **external** documentation.

drawTransformFeedbackStreamInstanced(Mode, Id, Stream, Primcount) -> ok

Types:

```
Mode = enum()
Id = integer()
Stream = integer()
Primcount = integer()
```

glDrawTransformFeedbackStreamInstance

See **external** documentation.

getInternalformativ(Target, Internalformat, Pname, BufSize) -> [integer()]

Types:

```
Target = enum()
Internalformat = enum()
Pname = enum()
BufSize = integer()
```

glGetInternalformat

See **external** documentation.

bindImageTexture(Unit, Texture, Level, Layered, Layer, Access, Format) -> ok

Types:

```
Unit = integer()
Texture = integer()
Level = integer()
Layered = 0 | 1
Layer = integer()
Access = enum()
Format = enum()
```

Bind a level of a texture to an image unit

`gl:bindImageTexture` binds a single level of a texture to an image unit for the purpose of reading and writing it from shaders. `Unit` specifies the zero-based index of the image unit to which to bind the texture level. `Texture` specifies the name of an existing texture object to bind to the image unit. If `Texture` is zero, then any existing binding to the image unit is broken. `Level` specifies the level of the texture to bind to the image unit.

If `Texture` is the name of a one-, two-, or three-dimensional array texture, a cube map or cube map array texture, or a two-dimensional multisample array texture, then it is possible to bind either the entire array, or only a single layer of the array to the image unit. In such cases, if `Layered` is `?GL_TRUE`, the entire array is attached to the image unit and `Layer` is ignored. However, if `Layered` is `?GL_FALSE` then `Layer` specifies the layer of the array to attach to the image unit.

Access specifies the access types to be performed by shaders and may be set to ?GL_READ_ONLY , ?GL_WRITE_ONLY, or ?GL_READ_WRITE to indicate read-only, write-only or read-write access, respectively. Violation of the access type specified in Access (for example, if a shader writes to an image bound with Access set to ?GL_READ_ONLY) will lead to undefined results, possibly including program termination.

Format specifies the format that is to be used when performing formatted stores into the image from shaders. Format must be compatible with the texture's internal format and must be one of the formats listed in the following table.

Image Unit Format	Format Qualifier
?GL_RGBA32F	rgba32f
?GL_RGBA16F	rgba16f
?GL_RG32F	rg32f
?GL_RG16F	rg16f
?GL_R11F_G11F_B10F	r11f_g11f_b10f
?GL_R32F	r32f
?GL_R16F	r16f
?GL_RGBA32UI	rgba32ui
?GL_RGBA16UI	rgba16ui
?GL_RGB10_A2UI	rgb10_a2ui
?GL_RGBA8UI	rgba8ui
?GL_RG32UI	rg32ui
?GL_RG16UI	rg16ui
?GL_RG8UI	rg8ui
?GL_R32UI	r32ui
?GL_R16UI	r16ui
?GL_R8UI	r8ui
?GL_RGBA32I	rgba32i
?GL_RGBA16I	rgba16i
?GL_RGBA8I	rgba8i
?GL_RG32I	rg32i
?GL_RG16I	rg16i
?GL_RG8I	rg8i
?GL_R32I	r32i
?GL_R16I	r16i
?GL_R8I	r8i
?GL_RGBA16	rgba16
?GL_RGB10_A2	rgb10_a2
?GL_RGBA8	rgba8
?GL_RG16	rg16
?GL_RG8	rg8
?GL_R16	r16
?GL_R8	r8
?GL_RGBA16_SNORM	rgba16_snorm
?GL_RGBA8_SNORM	rgba8_snorm
?GL_RG16_SNORM	rg16_snorm
?GL_RG8_SNORM	rg8_snorm
?GL_R16_SNORM	r16_snorm
?GL_R8_SNORM	r8_snorm

When a texture is bound to an image unit, the Format parameter for the image unit need not exactly match the texture internal format as long as the formats are considered compatible as defined in the OpenGL Specification. The matching criterion used for a given texture may be determined by calling *gl:getTexParameterfv/2* with Value set to ?GL_IMAGE_FORMAT_COMPATIBILITY_TYPE,

with return values of ?GL_IMAGE_FORMAT_COMPATIBILITY_BY_SIZE and ?GL_IMAGE_FORMAT_COMPATIBILITY_BY_CLASS, specifying matches by size and class, respectively.

See **external** documentation.

memoryBarrier(Barriers) -> ok

Types:

Barriers = integer()

Defines a barrier ordering memory transactions

gl:memoryBarrier defines a barrier ordering the memory transactions issued prior to the command relative to those issued after the barrier. For the purposes of this ordering, memory transactions performed by shaders are considered to be issued by the rendering command that triggered the execution of the shader. Barriers is a bitfield indicating the set of operations that are synchronized with shader stores; the bits used in Barriers are as follows:

?GL_VERTEX_ATTRIB_ARRAY_BARRIER_BIT: If set, vertex data sourced from buffer objects after the barrier will reflect data written by shaders prior to the barrier. The set of buffer objects affected by this bit is derived from the buffer object bindings used for generic vertex attributes derived from the ?GL_VERTEX_ATTRIB_ARRAY_BUFFER bindings.

?GL_ELEMENT_ARRAY_BARRIER_BIT: If set, vertex array indices sourced from buffer objects after the barrier will reflect data written by shaders prior to the barrier. The buffer objects affected by this bit are derived from the ?GL_ELEMENT_ARRAY_BUFFER binding.

?GL_UNIFORM_BARRIER_BIT: Shader uniforms sourced from buffer objects after the barrier will reflect data written by shaders prior to the barrier.

?GL_TEXTURE_FETCH_BARRIER_BIT: Texture fetches from shaders, including fetches from buffer object memory via buffer textures, after the barrier will reflect data written by shaders prior to the barrier.

?GL_SHADER_IMAGE_ACCESS_BARRIER_BIT: Memory accesses using shader image load, store, and atomic built-in functions issued after the barrier will reflect data written by shaders prior to the barrier. Additionally, image stores and atomics issued after the barrier will not execute until all memory accesses (e.g., loads, stores, texture fetches, vertex fetches) initiated prior to the barrier complete.

?GL_COMMAND_BARRIER_BIT: Command data sourced from buffer objects by Draw*Indirect commands after the barrier will reflect data written by shaders prior to the barrier. The buffer objects affected by this bit are derived from the ?GL_DRAW_INDIRECT_BUFFER binding.

?GL_PIXEL_BUFFER_BARRIER_BIT: Reads and writes of buffer objects via the ?GL_PIXEL_PACK_BUFFER and ?GL_PIXEL_UNPACK_BUFFER bindings (via *gl:readPixels/7* , *gl:texSubImage1D/7* , etc.) after the barrier will reflect data written by shaders prior to the barrier. Additionally, buffer object writes issued after the barrier will wait on the completion of all shader writes initiated prior to the barrier.

?GL_TEXTURE_UPDATE_BARRIER_BIT: Writes to a texture via gl:tex(Sub)Image*, gl:copyTex(Sub)Image* , gl:compressedTex(Sub)Image*, and reads via *gl:getTexImage/5* after the barrier will reflect data written by shaders prior to the barrier. Additionally, texture writes from these commands issued after the barrier will not execute until all shader writes initiated prior to the barrier complete.

?GL_BUFFER_UPDATE_BARRIER_BIT: Reads or writes via *gl:bufferSubData/4* , *gl:copyBufferSubData/5* , or *gl:getBufferSubData/4* , or to buffer object memory mapped by see glMapBuffer or see glMapBufferRange after the barrier will reflect data written by shaders prior to the barrier. Additionally, writes via these commands issued after the barrier will wait on the completion of any shader writes to the same memory initiated prior to the barrier.

?GL_FRAMEBUFFER_BARRIER_BIT: Reads and writes via framebuffer object attachments after the barrier will reflect data written by shaders prior to the barrier. Additionally, framebuffer writes issued after the barrier will wait on the completion of all shader writes issued prior to the barrier.

?GL_TRANSFORM_FEEDBACK_BARRIER_BIT: Writes via transform feedback bindings after the barrier will reflect data written by shaders prior to the barrier. Additionally, transform feedback writes issued after the barrier will wait on the completion of all shader writes issued prior to the barrier.

?GL_ATOMIC_COUNTER_BARRIER_BIT: Accesses to atomic counters after the barrier will reflect writes prior to the barrier.

If `Barriers` is ?GL_ALL_BARRIER_BITS, shader memory accesses will be synchronized relative to all the operations described above.

Implementations may cache buffer object and texture image memory that could be written by shaders in multiple caches; for example, there may be separate caches for texture, vertex fetching, and one or more caches for shader memory accesses. Implementations are not required to keep these caches coherent with shader memory writes. Stores issued by one invocation may not be immediately observable by other pipeline stages or other shader invocations because the value stored may remain in a cache local to the processor executing the store, or because data overwritten by the store is still in a cache elsewhere in the system. When `gl:memoryBarrier` is called, the GL flushes and/or invalidates any caches relevant to the operations specified by the `Barriers` parameter to ensure consistent ordering of operations across the barrier.

To allow for independent shader invocations to communicate by reads and writes to a common memory address, image variables in the OpenGL Shading Language may be declared as "coherent". Buffer object or texture image memory accessed through such variables may be cached only if caches are automatically updated due to stores issued by any other shader invocation. If the same address is accessed using both coherent and non-coherent variables, the accesses using variables declared as coherent will observe the results stored using coherent variables in other invocations. Using variables declared as "coherent" guarantees only that the results of stores will be immediately visible to shader invocations using similarly-declared variables; calling `gl:memoryBarrier` is required to ensure that the stores are visible to other operations.

The following guidelines may be helpful in choosing when to use coherent memory accesses and when to use barriers.

Data that are read-only or constant may be accessed without using coherent variables or calling MemoryBarrier(). Updates to the read-only data via API calls such as BufferSubData will invalidate shader caches implicitly as required.

Data that are shared between shader invocations at a fine granularity (e.g., written by one invocation, consumed by another invocation) should use coherent variables to read and write the shared data.

Data written by one shader invocation and consumed by other shader invocations launched as a result of its execution ("dependent invocations") should use coherent variables in the producing shader invocation and call memoryBarrier() after the last write. The consuming shader invocation should also use coherent variables.

Data written to image variables in one rendering pass and read by the shader in a later pass need not use coherent variables or memoryBarrier(). Calling MemoryBarrier() with the SHADER_IMAGE_ACCESS_BARRIER_BIT set in `Barriers` between passes is necessary.

Data written by the shader in one rendering pass and read by another mechanism (e.g., vertex or index buffer pulling) in a later pass need not use coherent variables or memoryBarrier(). Calling `gl:memoryBarrier` with the appropriate bits set in `Barriers` between passes is necessary.

See **external** documentation.

texStorage1D(Target, Levels, Internalformat, Width) -> ok

Types:

```
Target = enum()
Levels = integer()
Internalformat = enum()
Width = integer()
```

Simultaneously specify storage for all levels of a one-dimensional texture

gl:texStorage1D specifies the storage requirements for all levels of a one-dimensional texture simultaneously. Once a texture is specified with this command, the format and dimensions of all levels become immutable unless it is a proxy texture. The contents of the image may still be modified, however, its storage requirements may not change. Such a texture is referred to as an immutable-format texture.

Calling gl:texStorage1D is equivalent, assuming no errors are generated, to executing the following pseudo-code: for (i = 0; i < levels; i++) { glTexImage1D(target, i, internalformat, width, 0, format, type, NULL); width = max(1, (width / 2)); }

Since no texture data is actually provided, the values used in the pseudo-code for Format and Type are irrelevant and may be considered to be any values that are legal for the chosen Internalformat enumerant. Internalformat must be one of the sized internal formats given in Table 1 below, one of the sized depth-component formats ?GL_DEPTH_COMPONENT32F , ?GL_DEPTH_COMPONENT24, or ?GL_DEPTH_COMPONENT16, or one of the combined depth-stencil formats, ?GL_DEPTH32F_STENCIL8, or ?GL_DEPTH24_STENCIL8. Upon success, the value of ?GL_TEXTURE_IMMUTABLE_FORMAT becomes ?GL_TRUE. The value of ?GL_TEXTURE_IMMUTABLE_FORMAT may be discovered by calling *gl:getTexParameterfv/2* with Pname set to ?GL_TEXTURE_IMMUTABLE_FORMAT. No further changes to the dimensions or format of the texture object may be made. Using any command that might alter the dimensions or format of the texture object (such as *gl:texImage1D/8* or another call to gl:texStorage1D) will result in the generation of a ?GL_INVALID_OPERATION error, even if it would not, in fact, alter the dimensions or format of the object.

See **external** documentation.

texStorage2D(Target, Levels, Internalformat, Width, Height) -> ok

Types:

Target = enum()
Levels = integer()
Internalformat = enum()
Width = integer()
Height = integer()

Simultaneously specify storage for all levels of a two-dimensional or one-dimensional array texture

gl:texStorage2D specifies the storage requirements for all levels of a two-dimensional texture or one-dimensional texture array simultaneously. Once a texture is specified with this command, the format and dimensions of all levels become immutable unless it is a proxy texture. The contents of the image may still be modified, however, its storage requirements may not change. Such a texture is referred to as an immutable-format texture.

The behavior of gl:texStorage2D depends on the Target parameter. When Target is ?GL_TEXTURE_2D, ?GL_PROXY_TEXTURE_2D, ?GL_TEXTURE_RECTANGLE, ?GL_PROXY_TEXTURE_RECTANGLE or ?GL_PROXY_TEXTURE_CUBE_MAP, calling gl:texStorage2D is equivalent, assuming no errors are generated, to executing the following pseudo-code: for (i = 0; i < levels; i++) { glTexImage2D(target, i, internalformat, width, height, 0, format, type, NULL); width = max(1, (width / 2)); height = max(1, (height / 2)); }

When Target is ?GL_TEXTURE_CUBE_MAP, gl:texStorage2D is equivalent to: for (i = 0; i < levels; i++) { for (face in (+X, -X, +Y, -Y, +Z, -Z)) { glTexImage2D(face, i, internalformat, width, height, 0, format, type, NULL); } width = max(1, (width / 2)); height = max(1, (height / 2)); }

When Target is ?GL_TEXTURE_1D or ?GL_TEXTURE_1D_ARRAY, gl:texStorage2D is equivalent to: for (i = 0; i < levels; i++) { glTexImage2D(target, i, internalformat, width, height, 0, format, type, NULL); width = max(1, (width / 2)); }

Since no texture data is actually provided, the values used in the pseudo-code for Format and Type are irrelevant and may be considered to be any values that are legal for the chosen Internalformat enumerant. Internalformat must be one of the sized internal formats given in Table 1 below, one of the sized depth-component formats ?GL_DEPTH_COMPONENT32F , ?GL_DEPTH_COMPONENT24, or ?GL_DEPTH_COMPONENT16, or

one of the combined depth-stencil formats, ?GL_DEPTH32F_STENCIL8, or ?GL_DEPTH24_STENCIL8. Upon success, the value of ?GL_TEXTURE_IMMUTABLE_FORMAT becomes ?GL_TRUE. The value of ?GL_TEXTURE_IMMUTABLE_FORMAT may be discovered by calling *gl:getTexParameterfv/2* with Pname set to ?GL_TEXTURE_IMMUTABLE_FORMAT. No further changes to the dimensions or format of the texture object may be made. Using any command that might alter the dimensions or format of the texture object (such as *gl:texImage2D/9* or another call to gl:texStorage2D) will result in the generation of a ?GL_INVALID_OPERATION error, even if it would not, in fact, alter the dimensions or format of the object.

See **external** documentation.

texStorage3D(Target, Levels, Internalformat, Width, Height, Depth) -> ok

Types:

Target = enum()
Levels = integer()
Internalformat = enum()
Width = integer()
Height = integer()
Depth = integer()

Simultaneously specify storage for all levels of a three-dimensional, two-dimensional array or cube-map array texture

gl:texStorage3D specifies the storage requirements for all levels of a three-dimensional, two-dimensional array or cube-map array texture simultaneously. Once a texture is specified with this command, the format and dimensions of all levels become immutable unless it is a proxy texture. The contents of the image may still be modified, however, its storage requirements may not change. Such a texture is referred to as an immutable-format texture.

The behavior of gl:texStorage3D depends on the Target parameter. When Target is ?GL_TEXTURE_3D, or ?GL_PROXY_TEXTURE_3D, calling gl:texStorage3D is equivalent, assuming no errors are generated, to executing the following pseudo-code: for (i = 0; i < levels; i++) { glTexImage3D(target, i, internalformat, width, height, depth, 0, format, type, NULL); width = max(1, (width / 2)); height = max(1, (height / 2)); depth = max(1, (depth / 2)); }

When Target is ?GL_TEXTURE_2D_ARRAY, ?GL_PROXY_TEXTURE_2D_ARRAY, ?GL_TEXTURE_CUBE_MAP_ARRAY , or ?GL_PROXY_TEXTURE_CUBE_MAP_ARRAY, gl:texStorage3D is equivalent to: for (i = 0; i < levels; i++) { glTexImage3D(target, i, internalformat, width, height, depth, 0, format, type, NULL); width = max(1, (width / 2)); height = max(1, (height / 2)); }

Since no texture data is actually provided, the values used in the pseudo-code for Format and Type are irrelevant and may be considered to be any values that are legal for the chosen Internalformat enumerant. Internalformat must be one of the sized internal formats given in Table 1 below, one of the sized depth-component formats ?GL_DEPTH_COMPONENT32F , ?GL_DEPTH_COMPONENT24, or ?GL_DEPTH_COMPONENT16, or one of the combined depth-stencil formats, ?GL_DEPTH32F_STENCIL8, or ?GL_DEPTH24_STENCIL8. Upon success, the value of ?GL_TEXTURE_IMMUTABLE_FORMAT becomes ?GL_TRUE. The value of ?GL_TEXTURE_IMMUTABLE_FORMAT may be discovered by calling *gl:getTexParameterfv/2* with Pname set to ?GL_TEXTURE_IMMUTABLE_FORMAT. No further changes to the dimensions or format of the texture object may be made. Using any command that might alter the dimensions or format of the texture object (such as *gl:texImage3D/10* or another call to gl:texStorage3D) will result in the generation of a ?GL_INVALID_OPERATION error, even if it would not, in fact, alter the dimensions or format of the object.

See **external** documentation.

depthBoundsEXT(Zmin, Zmax) -> ok

Types:

Zmin = clamp()

Zmax = clamp()

glDepthBoundsEXT

See **external** documentation.

stencilClearTagEXT(StencilTagBits, StencilClearTag) -> ok

Types:

StencilTagBits = integer()

StencilClearTag = integer()

glStencilClearTagEXT

See **external** documentation.